Enabling Post-Quantum Signatures in DNSSEC: One ARRF at a time

by

Jason Goertzen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The Domain Name System Security Extensions (DNSSEC) provide authentication of DNS responses using digital signatures. DNS relies on UDP as its primary delivery method which imposes several constraints, with the most notable being that DNS message sizes should be no larger than 1232 bytes to avoid message delivery issues. It is possible to deliver larger DNS messages by either utilizing UDP fragmentation or falling back to TCP, but neither are sufficiently reliable in the current DNS ecosystem. Although large DNSSEC messages are not a primary concern today — due to the signature size of actively used algorithms such as RSA or elliptic curve cryptography — large DNS messages become an alarming issue for post-quantum signing algorithms due to their larger signatures and/or keys. In this thesis, we propose ARRF, a method for fragmenting large DNS resource records at the application layer (rather than the transport layer). ARRF is a *request-based* fragmentation method, meaning that the initial response contains a truncated response and all remaining fragments must be explicitly requested. By using request-based fragmentation, ARRF avoids issues of previously proposed—and rejected—application-layer DNS fragmentation techniques. By requiring fragments to be explicitly requested at the application layer we avoid issues caused by problematic network devices along the transmission path.

We implement ARRF and evaluate its performance on a simulated network when used for the three post-quantum algorithms selected by NIST for standardization (Falcon, CRYSTALS-Dilithium and SPHINCS+) at the 128-bit security level. Our experiments show that ARRF has considerably lower resolution times compared to DNS over UDP with TCP fallback for all tested algorithms. We also find that, when using ARRF to deliver Falcon and Dilithium less data transmission is required. ARRF was also designed with a low implementation burden. Our implementation is a simple lightweight daemon which sits in front of DNS name servers and resolvers and performs the fragmentation and reassembly transparently.

Acknowledgements

I would first like to thank my supervisor Dr Douglas Stebila for all the time and mentorship he provided me while working on this thesis and for taking a chance on me. Supervising a student halfway across the country due to a global pandemic must have been difficult, but he made it look easy. Douglas went above and beyond with his guidence, patience, and care. Not only did he care about me as a student but also as a person. Thank you.

I would also like to thank my thesis readers, Dr Urs Hengartner, Dr Ian Goldberg, and Dr Alfred Menezes, for their valuable time and feedback.

I would not be in the position I am in without the support and mentorship that Dr Nathaniel Osgood gave me during my undergraduate research. He not only provided but embraced the opportunity to do research in cryptography despite not being primary research topic of his in the past. I would also like to thank Dr Cameron Franc and Dr Christopher Duffy for encouraging me to pursue math during my undergraduate degree. Thank you all.

The entire CrySP lab has been such a welcoming and uplifting community and is genuinely one of the highlights of my time here at Waterloo. I want to specifically thank Thomas, Rasoul, Lucas, John, Simon, Ted, Adithya, Shannon, Vasisht, Emily, Bailey, Miti, Jarrod, and Nils. Despite essentially being remote for almost two years, you all made me feel welcome and a part of the CrySP community. I would also like to thank Camryn for putting up with my thesis ranting and slow pace on our runs together.

I also want to thank Colin, Caylee, Frank, Chris, Mike, Derek, Ben, Brian, Trevor, Tyler, Lauressa, Aaron and Robin. Graduate school can take a massive toll on your mental health; without all of you, I would probably have gone completely insane.

Finally, I want to thank my loving parents, Darrell and Sherry, and late grandmother, Evelyn, for their love and support all these years. I could not have done any of this without you.

Dedication

To my parents, Darrell, and Sherry, and my late Grandmother Evelyn. Thank you for all of the love and support over the years.

Table of Contents

List of Figures viii

List of Tables ix

1 Introduction 1

2 Background on the Domain Name System 4

2.1 The Domain Name System 4

2.2 Security Concerns for DNS 5

2.3 Domain Name Security Extensions 7

2.3.1 Other DNS Security Mechanisms 9

2.4 DNSSEC Challeneges 10

2.4.1 Using DNSSEC for Amplification Denial of Service Attacks 10

2.4.2 UDP/IP Fragmentation and DNSSEC 12

2.4.3 DNSSEC Configuration 16

2.4.4 DNSSEC Adoption 17

2.5 DNSSEC Algorithm Life Cycle 18

2.6 Root zone key rollover 19

3 Post-quantum Cryptography 21

3.1 NIST Post-Quantum Cryptography Selection Process 21

3.2 Selected Algorithms and Round 4 Candidates . . . 23
3.2.1 Digital signature algorithms . . . 23
3.2.2 Public-key encryption/key encapsulation mechanisms . . . 25
3.3 Stateful Hash-Based Signatures . . . 28
3.4 Post-Quantum Cryptography and the Domain Name System . . . 28

4 Adapting DNSSEC for a Quantum Future **31**
4.1 Request based fragmentation . . . 31
4.1.1 Resource Record Fragments . . . 32
4.1.2 Using RRFrags . . . 33
4.1.3 Example execution of ARRF . . . 34
4.1.4 Caching and DNSSEC Considerations . . . 36
4.2 Evaluating Post-quantum DNSSEC . . . 36
4.2.1 Algorithm performance . . . 38
4.2.2 Post-quantum with standard DNSSEC . . . 38
4.2.3 Post-quantum with ARRF . . . 39
4.2.4 Post-quantum data transmission . . . 40
4.2.5 Results . . . 40
4.3 Discussion . . . 49
4.3.1 ARRF Performance . . . 49
4.3.2 ARRF Backwards Compatibility . . . 49
4.3.3 ARRF Security Considerations . . . 50
4.3.4 Comparing ARRF against previously proposed mechanisms . . . 51

5 Future Work and Conclusion **54**
5.1 Future Work . . . 54
5.2 Conclusion . . . 55

References **57**

List of Figures

2.1 A simplified version of the DNS distributed tree. 5

2.2 An illustration of a message larger than the MTU size being split into two fragments. The second fragment does not contain the UDP header 13

4.1 The mapping of the RRFrag format onto the generic resource record format. 33

4.2 A simplified example execution of ARRF 35

4.3 Resolution times in milliseconds with 10ms delay and 128 Kilobytes per second bandwidth 44

4.4 Resolution times in milliseconds with 10ms delay and 50 Megabytes per second bandwidth 45

4.5 Resolution times in milliseconds with 10ms delay and 50 Megabytes per second bandwidth 46

4.6 Resolution times in milliseconds with 0ms delay and unlimited bandwidth . 47

List of Tables

3.1 NIST Security levels 22
3.2 CRYSTALS-Dilithium's parameter sets. 23
3.3 Falcon's parameter sets 24
3.4 SPHINCS+'s parameter sets. 25
3.5 CRYSTALS-KYBER's parameter sets. 26
3.6 BIKE's parameter sets. 26
3.7 Classic McEliece's parameter sets. 27
3.8 HQC's parameter sets. 27
3.9 SIKE's parameter sets. 28

4.1 Algorithm runtime measured using OpenSSL Speed 38
4.2 Resolution times plus or minus standard deviation without ARRF 39
4.3 Resolution times plus and minus standard deviation with the sequential ARRF daemon 41
4.4 Resolution times plus and minus standard deviation with the Parallel ARRF daemon 42
4.5 Total data transmitted when performing a DNS lookup between resolver and name server 42

Chapter 1

Introduction

The Domain Name System (DNS) is a mission critical service for the Internet. DNS is responsible for translating human-readable domain names into machine-understandable IP addresses and is used by billions of devices daily. Ensuring that these translations are correct and not forged is critical to prevent users from being directed to malicious servers instead of their intended destination. The Domain Name System Security Extensions (DNSSEC) [57] provide data integrity by using digital signatures. DNSSEC ensures that the received DNS message is indeed from a server authorized to respond to the query, and that the message has not been modified in transit.

Today's DNSSEC uses digital signatures that rely on traditional security assumptions such as factoring and discrete logarithms, which would not resist attacks by a cryptographically relevant quantum computer. To continue to provide its intended security guarantees in the face of such threats, DNSSEC must be updated to accommodate quantum-resistant algorithms. The post-quantum cryptography standardization project of the United States National Institute of Standards and Technology (NIST) announced in July 2022 [3] three post-quantum digital signatures algorithms to be standardized: CRYSTALS-Dilithium [10], Falcon [28], and SPHINCS+ [8]. All of these selected algorithms have one thing in common: the amount of data transmission required in order to perform a verification is substantially larger than their non-post-quantum counterparts: both public keys and signatures. This increase in size can cause substantial issues for pre-existing network protocols; DNS and DNSSEC are particularly sensitive to this issue.

DNS uses User Datagram Protocol (UDP) as its primary delivery protocol which allows for at most 1500 bytes to be transmitted per packet before fragmentation. Although this does not seem like a major issue, UDP fragmentation is quite fragile and can often cause

delivery issues. Once the various headers are accounted for, DNS can send messages of at most 1232 bytes before needing to fall back to TCP which does not suffer from the same fragmentation issues. Unfortunately, these post-quantum signatures can cause DNS messages to exceed the 1232 byte UDP limit, and a non-trivial amount of the DNS does not support TCP.

In this work we propose 'A Resource Record Fragmentation mechanism', or 'ARRF' for short. ARRF is a lightweight request-based backwards compatible fragmentation mechanism which entirely relies on UDP and keeps all DNS communication below the 1232 byte limit. ARRF works by moving DNS message fragmentation from UDP to the DNS level itself. Large resource records are individually fragmented and fragments are only sent when explicitly requested. ARRF is also designed in such a way that it can be implemented with low impact on existing servers; in fact we were able to implement it as a transparent daemon sitting in front of an ARRF-unaware requester and resolver at both ends of a DNS lookup request, reducing the burden of deployment. To evaluate our approach, we implemented the three post-quantum digital signature algorithms selected by NIST — specifically, parameter sets Falcon-512, Dilithium2, and SPHINCS+-SHA256-128s— in BIND using liboqs [61], as well as a daemon implementing ARRF sitting in front of the requester and resolver, transparently carrying out the ARRF fragmentation/reassembly. We were then able to carry out a variety of experiments on a simulated network with different latencies and bandwidth and different fragmentation sizes to evaluate the performance of ARRF compared to DNS over UDP with TCP fallback, measuring the total resolution time and the amount of data transmitted. In all our tested scenarios, we found that Falcon-512 performs better than both Dilithium2 and SPHINCS+-SHA256-128s due to Falcon-512's smaller signatures, suggesting that Falcon-512 may be the most suitable option currently available to be standardized for DNSSEC. We did however find that even with the improved performance of post-quantum algorithms in ARRF compared to standard DNS over UDP with TCP fallback, post-quantum algorithms incurred a performance penalty compared to non-post-quantum algorithms currently in use with DNSSEC (RSA and ECDSA) due to the unavoidable cost of transmitting more data. Overall, we conclude that ARRF is a promising option for transitioning to post-quantum DNSSEC: it has less performance degradation compared to standard DNS over UDP with TCP fallback.

The rest of the thesis is organized as follows. In Chapter 2, we introduce the Domain Name System and its security concerns. We then provide an introduction to DNSSEC and the challenges it faces, such as UDP/IP Fragmentation. We also discuss the process of standardizing a new algorithm for DNSSEC and the process of changing the root zone's keys.

Chapter 3 introduces post-quantum cryptography and why it is important. We then

provide a brief history of NIST's post-quantum selection process up to the conclusion of the third round. Finally, we introduce all of the algorithms which were either selected for standardization or selected to continue into round 4 of the selection process.

In Chapter 4, we introduce the ARRF design. We use a lightweight proof of concept daemon to evaluate the effectiveness of ARRF at delivering DNSSEC messages using post-quantum signatures. We also provide an experimental evaluatation comparing how Falcon, CRYSTALS-Dilithim, and SPHINCS+ perform both with and without ARRF and provide a comparison against currently deployed DNSSEC algorithms RSA and ECDSA.

Finally, Chapter 5 outlines the additional work required before deploying ARRF.

Chapter 2

Background on the Domain Name System

DNS is a critical piece of internet infrastructure. In this chapter, we describe how DNS works, and threats against DNS. We then discuss the mechanism designed to prevent those attacks, DNSSEC. We also discuss the challenges DNSSEC faces with delivery, deployment, configuration, and adoption. We conclude the chapter by discussing the process of getting a new algorithm standardized and deployed to DNSSEC and the process of changing the root zone's keys.

2.1 The Domain Name System

Computers use IP addresses to address messages to each other. The DNS provides a mechanism to translate between human readable strings and these IP addresses. Originally conceptualized in 1983 in RFC 882 [48] and RFC 883 [49], DNS provides a distributed tree of *zones* which may or may not be on different machines. Each zone will contain a set of *Resource Records* (RRs) with differing types, and labels. By breaking up DNS into zones, it improves the scalability of the systems by allowing individuals to manage their own zones and name servers, while reducing the strain on a particular machine by delegating responses to other zones and using caching. To assist in the explanation, we will use the situation where a user wants to access *uwaterloo.ca*, as illustrated in Figure 2.1. Assuming the requested address is not cached, the DNS resolver will contact the *root* name servers to ask which Top Level Domain (TLD) name server contains information

about *.ca* domain names. The root name servers will then respond with *NS* and *A/AAAA* records which indicate the name server responsible for the *.ca* zone. Once the resolver receives a reply, it would then send a request to the *.ca* TLD name servers asking for the NS and A/AAAA records associated with the Second Level Domain (SLD) *uwaterloo.ca.* The TLD name servers would then reply with the records for the name server responsible for the *uwaterloo.ca* zone. Finally, the resolver sends a request to the uwaterloo name server for the *A* or *AAAA* record associated with the IP address of *uwaterloo.ca.* Although in our example case the translation would stop at the uwaterloo.ca level, there may be additional name servers required. Due to the amount of requests made, caching is used to reduce the strain on name servers as well as reducing latency.

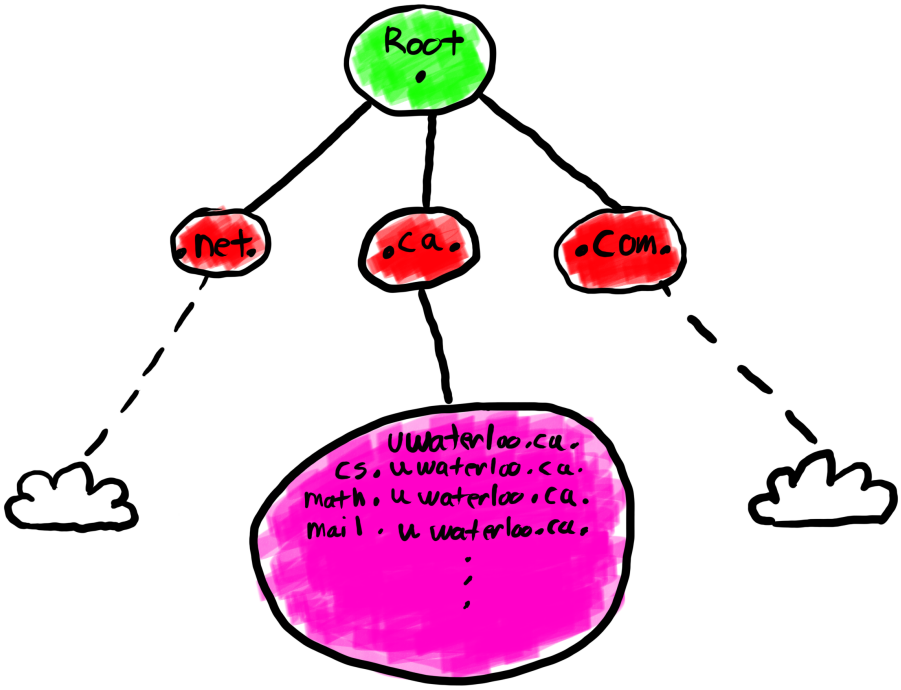


Figure 2.1: A simplified version of the DNS distributed tree.

2.2 Security Concerns for DNS

DNS as originally designed contains no guarantees on security, and no notion of cryptographic authentication, leaving the protocol susceptible to attacks. The Internet Engi-

neering Task Force (IETF) decided there was a need to identify these threats to DNS and published RFC 3833 [7] which describes weaknesses that can be exploited by malicious actors for both DNS and DNSSEC

The primary class of threats involve poisoning a target DNS resolver's cache, which can be done a few different ways:

i) Packet interception, also known as a Monkey-in-the-Middle attack, is an attack where a malicious actor is able to sit in the chain of communication between a source and destination, and allows the malicious actor to not just see what is communicated between the two servers, but the malicious actor can also modify, drop, or pass on packets that are part of that communication. This allows the malicious actor to see what a server is requesting, and provide a malicious answer. In our context, the attacker would see a DNS lookup request and reply with an IP address for a malicious server rather than the server the DNS resolver is actually looking for; this could then get cached by said DNS resolver.

ii) ID Guessing and Query Prediction is essentially a brute force attack where the malicious actor constructs UDP packets and tries to make them look like valid replies from a name server so that malicious resource records are cached. In order to do this, the adversary must be able to guess the question being asked, and the ID field.[1] Considering the ID field in a DNS header is only 16 bits long, and the server port used for DNS is public information, there are only 2^{16} possible combinations to try. In 2008 Kaminsky [40] showed that you can reliably poison a DNS resolver's cache by querying for records which are not in the zone, and then flooding the resolver with responses that delegrate the response to a malicious server. In response to this attack randomized client ports were introduced thus effectively increasing the difficulty of guessing to 1 in 2^{32}. In 2021, Man et al. [44] introduced a new technique for infering the client's UDP source port. This technique reduces the search space to $2^{16} + 2^{16}$. If the ID and source port can sucessfully guessed then a malicious record could then be cached.

These issues are alarming as they can be abused to perform denial of service and phishing attacks.

[1] An ID field is used to help the resolver determine which request is getting serviced.

2.3 Domain Name Security Extensions

To combat the threats described in Section 2.2, the IETF began drafting DNSSEC with the original proposal coming in 1999 as RFC 2535 [24] and was superseded in 2005 by RFC 4033 [57]. A important thing to bring up early when talking about DNSSEC is DNSSEC is not a confidentiality solution, it is designed solely as an integrity solution. That is to say, a malicious actor can still see what is being requested, by whom, and the reply, but if they modify or replace a reply with anything else that the replying name server did not include in the response the resolver should be able to detect it.

DNSSEC's integrity checks rely on constructing a *Chain of Trust* from the root zone, to the TLD zone, to the SLD zone, all the way until the answer to the query is found with each step being verified using various signature schemes. DNSSEC is flexible in that it supports multiple algorithms some of which are optional and others which are required. DNSSEC also supports zones being signed with multiple different algorithms at a given time to maximize compatibility between a given zone and resolver. Responses now include the signatures (RRSIGs) of entire sets of records (RRsets) that share the same label and type being requested, the entire RRset rather than an individual record, as well as potentially the public keys (DNSKEYs) used to verify the RRSIGs, so responses are larger than standard DNS. Due to this extra information that is included in responses, DNSSEC is built on top of the EDNS0 [65]. EDNS0 is a set of mechanisms which extends the maximum supported DNS response size from 512 bytes to 64 kilobytes, and requires the addition of the pseudo-resource record 'OPT' in requests and responses. OPT records are not explicitly located in zones. Instead, OPT records are dynamically constructed when a DNS message is built. OPT records extend the return code field from 4 bits to 36 bits and provide space for an additional 16 flags. Finally, OPT also provides a dynamically sized field to pass key-value pairs known as "options".

Recall from Section 2.1 that DNS lookups generally take a recursive form. Going back to our uwaterloo.ca example, the resolver would query root for the .ca zone, and the root zone would reply with an IP address associated with the .ca zone. The resolver now needs to verify that a malicious actor has not modified this response. In order for a zone to sign a response, it must generate a public and private key pair known as the *zone signing key* pair. As part of the response from root, the standard information is sent along with a signature for each set of Resource Records (RRsets) being sent. These RRsets are signed by root's zone signing private key. If the resolver does not have root's zone signing public key, it must perform a DNSKEY query as well. The resolver will then use the root zones's zone signing public key with the signatures included in the response to verify the integrity of the response.

The reader may have noticed a potential problem with this mechanism as currently described: if the zone signing public key is provided via the same medium as the rest of the response, then there is nothing stopping a malicious actor from modifying the zone signing public key, the signature, and the response of the query before the resolver receives it. This undermines the entire point of DNSSEC. The way DNSSEC handles this issue is to include a signature of the zone signing public key signed by the private key of a second key pair known as the *key signing key* pair. DNSSEC does support the use of a single key pair for use as the zone signing key and key signing key, known as the *combined signing key* (CSK) pair, but it is far more common for there to be two separate key pairs; one for signing the zone and one for signing the zone's public key. The main obstacle with this mechanism is how to provide the key signing public key to the resolver without having the same issue as the zone signing public key. This key must be provided through an alternative mechanism. This is known as a *trust anchor* as it is where the chain of trust begins/where the chain is anchored.

This approach is not scalable for any zones other than root as there are too many zones to pre-install these key signing public keys for each one. Therefore, in order to continue this chain of trust, the *parent* zone, must also pass on the key signing public key of the child zone to the resolver. In our example, root would be the parent of the .ca zone. This entire process will continue until a zone is able to answer the query with the appropriate record. If any of the integrity checks fail, whether from a malicious actor, a misconfigured zone, or one of the zones in the chain not supporting DNSSEC, the query response can no longer be assumed to be authentic.

This process works if a query is for a record that exists, but DNSSEC must also be able to verify a response saying the request is querying something that does not exist is valid. If the resolver cannot verify this, then malicious actors can perform a denial of service attack by modifying the response to say the requested domain name does not exist when it does. RFC 6781 [41] describes the two mechanisms used to provide this service, NSEC and NSEC3.

The idea behind NSEC is that an authoritative name server will construct a chain of NSEC records for each domain name the name server is responsible for. This chain is sorted based on the cleartext of the records it represents. If a resolver queries for a record that does not exist, the name server will reply with the NSEC record which contains the record that would be just before the queried record in the chain and the record that would be just after. This gives the name server something to sign and proves to the resolver that the record they were querying does not exist in a way that can have its integrity verified. The fact that NSEC records are constructed using cleartext allows for zone enumeration which can allow an outside actor to discover all of the records stored in that zone easily.

Certain zone operators may find this undesirable, such as for regulatory reasons, so NSEC3 was created to make zone enumeration more computationally taxing. NSEC3 is largely the same as NSEC, however instead of using the cleartext of records when creating the NSEC record, a hashing algorithm with salt is used for obfuscation purposes. There is a variation of NSEC known as *white lies* where the name server constructs an NSEC record using false names that still allow the resolver to authenticate non-existence [68]. An example of this would be if a zone contained a.example.com, f.example.com, z.example.com, and a resolver asked for d.example.com. A valid white lies response could be an NSEC record containing b.example.com and e.example.com. Both of those addresses do not actually exist, but they still allow the resolver to prove non-existence.

2.3.1 Other DNS Security Mechanisms

DNSSEC is not the only DNS security mechanism standardized by the IETF. Some these other security mechanisms focus on confidentiality (DNS over TLS and DNS over HTTPS), some on authenticity (TSIG), and some on availability (DNS cookies).

DNS over TLS Standardized in 2016 [33], DNS over TLS (DoT) sends queries and responses using the Transport Layer Security protocol (TLS), with queries being sent to a new dedicated TCP port 853. This provides confidentiality of what is being queried, as well as message integrity. However, DoT only provides integrity and confidentiality between the client and resolver, whereas DNSSEC provides end-to-end integrity between the resolver and name servers. Due to using a new port, middle boxes and network operators will be able to identify that a DoT lookup is being performed.

DNS over HTTPS The concern of DoT traffic being identifiable spawned the idea of DNS over HTTPS (DoH). DNS messages are now sent using the HTTP protocol over TLS [31]. Queries are now sent to the standard HTTPS TCP port 443. By doing this, DoH is indistinguishable from standard HTTPS traffic, with the extra complexity of requiring the HTTP protocol.

Both of the above protocols should gain to benefit from the recently standardized *DNS over Dedicated QUIC* transport protocol [35].

TSIG Transaction signatures (TSIG) provides a method to authenticate that DNS messages are from a specific requester by using a shared symmetric secret key and a one-way

hash function [26]. TSIGs are generally used when a primary name server wishes to inform secondary name servers that they need to update their copy of a zone to match that of the primary name server, however they can also be used to ensure requests and responses are from authorized entities.

DNS Cookies Out of path attacks are a concern in the context of DNS, and cookies are a low cost protection to prevent these attacks [25]. When a DNS client makes a request, it will construct a 64 bit client cookie using a pseudo random function applied to the client's IP Address, the server's IP Address, and some secret value. When the server receives the request with the client cookie it will construct its 64 bit server cookie using a pseudo random function applied to the client's IP address, a its own secret value, and the client's cookie. When the response is sent both the server cookie and client cookie will be included. The client will then be able to compare the sent client cookie with what it expects. All future queries should include the client and server cookie. The server will then check the sent cookie against what it expects. If a cookie does not match what is expected, then a BADCOOKIE response is sent. This means that an out of path attacker must guess the correct number out of 2^{64} potential options.

2.4 DNSSEC Challeneges

Although DNSSEC achieves the goals that were set by the IETF, it also introduces some additional concerns. In this section, we discuss a few of these potential problems.

2.4.1 Using DNSSEC for Amplification Denial of Service Attacks

RFC 3833 [7] discusses that DNSSEC does not help against denial of service attacks; in fact it is pointed out that with DNSSEC deployed DNS can be used for larger scale denial of service amplification attacks.

Amplification attacks are attacks where a malicious actor spoofs a request and claims this spoofed request is from the victim server. The server receiving this request then replies with a larger amount of data than it received. DNSSEC worsens (or improves if you are the attacker) this amplification because of the extra information that has to be sent with response in order for a resolver to validate the integrity of that response.

Van Rijswijk-Deij et al. [64] sought to measure just how large of an impact DNSSEC would make for performing these types of attacks. As stock DNS can already be used for

amplification attacks, they used the theoretical maximum amplification of standard DNS to compare against the potential amplification against every DNSSEC enabled domain in the top 6 TLDs, and a measurement of sampled domains which do not have DNSSEC enabled in those same 6 TLDs.

They deduced that the theoretical maximum of standard DNS would occur when a request is made for the shortest domain possible (they use 'x.com' as an example) and the maximum allowed response size of 512 bytes, and results in an amplification of $\frac{512}{23} \approx 22.3$.

Comparing their measurements against each other and the theoretical max of standard DNS it was found that, perhaps not too surprisingly, DNSSEC has higher amplification than standard DNS. The interesting finding is that most DNSSEC query responses measured were within the theoretical maximum of standard DNS implying that DNSSEC may not be as bad as it might initially appear. It is found that the primary problem responses are 'ANY'[2] and 'DNSKEY' queries. Several DNS operators, including Cloudflare, have since deprecated ANY queries [43].

Van Rijswijk-Deij et al. proposed a few solutions to mitigate the observed increase in amplification.

i) Ingress Filtering: the primary goal of this solution is to prevent source address spoofing. With this solution, "network operators would only allow responses to enter their network if the source IP address is a legitimate address within their network." That is, if an attacker spoofs a source address that is not a legitimate address of their network, the network operator should drop the packet rather than forwarding it on. The problem with this proposal is that it requires that every network on the internet have Ingress Filtering deployed, and the sad reality is this is not the case, so this strategy will not work.

ii) Response Rate Limiting: the core idea behind Response Rate Limiting is for Authoritative Name Servers to limit the responses they send out if they receive a bunch of requests from the same IP address block. This solution was proposed after DNSSEC was used in a lot of attacks, and can be quite effective. However, this solution is not entirely helpful as the primary attack vector of DNSSEC amplification attacks are misconfigured resolvers which allow anyone to use them. If these resolvers cache the response from the authoritative name server, then they will be the root of the amplification and Response Rate Limiting is evaded.

[2] An 'ANY' query is a query which returns all records for a domain name.

iii) EDNS0 cookies: this proposed solution is Van Rijswijk-Deij et al.'s preferred solution and was originally proposed by Eastlake [25] . The idea is to only send large responses to queries that contain a valid cookie for the source IP address. This would greatly limit amplification attacks, but we note that this would add additional overhead to establishing this cookie. This overhead is not discussed in Van Rijswijk-Deij et al.'s work.

iv) Response size limiting: the core idea of response size limiting is to only send replies which would result in an amplification of less than the theoretical max of standard DNS. The authors theorize that likely only 'any' queries would be affected by this solution, and it would reduce the likelihood of fragmentation (see Section 2.4.2 for discussion on how fragmentation affects DNSSEC).

v) Restricting or blocking ANY queries: the authors note that Response Rate Limiting and response size limiting already restrict 'ANY' query responses so that they are not too large. However, flat out blocking ANY queries may end up breaking non-malicious software (such as qmail). It is also a bit of a narrow solution as queries such as 'DNSKEY' queries can also cause significant amplification. However, as previously mentioned, ANY queries are now no longer supported by several major DNS operators [43].

2.4.2 UDP/IP Fragmentation and DNSSEC

UDP/IP fragmentation

When sending messages in a network, message size becomes a concern. Different transport mediums have different maximum packet sizes known as the "Maximum transmission unit" (MTU) of the medium. If a desired message exceeds the MTU, it must be broken up and sent over multiple packets. Since TCP is a stream, this is handled gracefully and causes no routing issues. However, as UDP is datagram based, fragmentation can cause significant issues, with an IETF RFC commenting on its fragility [13].

When a message that is larger than the MTU is being sent over UDP the UDP packet is split into multiple fragments, each getting its own IP header.[3] The issue that arises with this is that the UDP header will only be in the first fragment as illustrated by Figure 2.2, thus resulting in the source and destination ports not being included in the other message

[3]IPv4 and IPv6 handle fragmentation slightly differently, but these differences are not important for this conversation

fragments. This can cause issues for middle boxes that use the source and destination ports to route packets for policy reasons. Stateless firewalls can also experience issues as the firewall must either accept all trailing fragments, which is susceptible to attack, or reject all trailing packets which can block legitimate traffic.

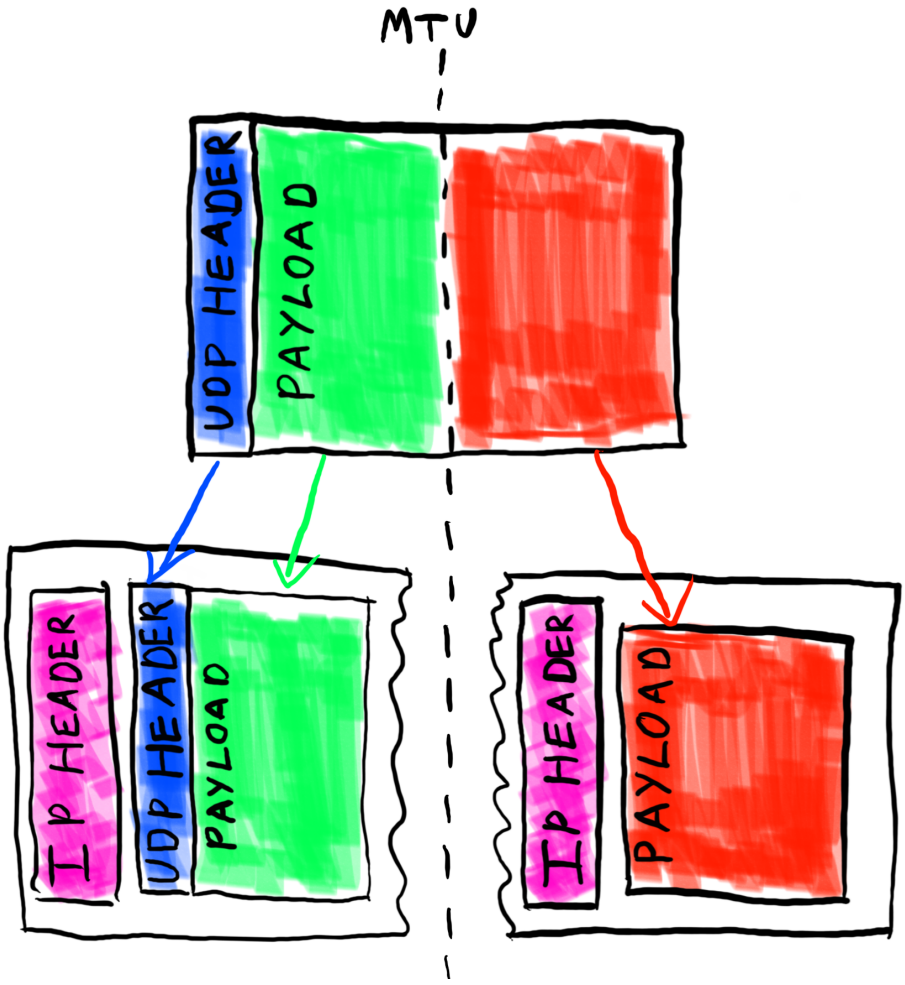


Figure 2.2: An illustration of a message larger than the MTU size being split into two fragments. The second fragment does not contain the UDP header

DNSSEC message fragmentation

DNSSEC can cause significantly larger messages to be sent compared to standard DNS due to the additional RRSIG and DNSKEY resource records sent in responses and can potentially require messages to be sent which are larger than a network path's MTU. Van Den Broek et al. [62] note that UDP/IP fragmentation issues do occur when using DNS, and they explore how widespread these issues are, if it is a concern for deploying DNSSEC, as well as propose a few potential solutions.

For their experiment, they collected network traces of an authoritative name server managed by SURFnet which is responsible for approximately 4000 zones, 300 of those having DNSSEC deployed. They found that 58% of queries received fragmented responses, with 10.5% of all resolvers seen showing problematic behaviour. They break the problematic behaviour up into five different resulting cases:

i) The resolver replies with an 'ICMP Fragment Reassembly Time Exceeded message' (ICMP FRTE). This occurs when a fragment gets dropped, most likely due to being blocked by a firewall. Approximately 1.3% of observed resolvers responded this way.

ii) The resolver sends a standard DNS request. This occurs when either EDNS0 is not supported by the name server being queried, or a fragment is dropped. A resolver cannot distinguish between the two, but the expect this error to occur due to a fragment being dropped. Approximately 2.4% of observed resolvers reverted to standard DNS.

iii) The resolver repeats the query but changes the EDNS0 advertised maximum response size which tells the name server the largest size the response can be.[4] Certain resolver software modifies this size if they are not able to reconstruct the response. Approximately 3.5% of observed resolvers retried with a modified advertised maximum response.

iv) The resolver reverts to using TCP to perform the DNSSEC lookup. This occurs when a resolver receives a truncated message, meaning that the name server was not able to fit the entire response into the advertised maximum size. Once TCP is being used as the method of communication, size is no longer a concern. It was found that communication was switched to TCP when the UDP based response was *not* truncated occurred less than 1% of the time.

[4] 69% of observed queries advertise 4096 bytes, 1.8% advertised 512 bytes and 2% advertised between 1280 bytes and 1472 bytes. Generally, the maximum size a UDP packet can be is 1500 bytes.

v) Some resolvers try the query again and receive large responses of over 512 bytes. The authors state that it is difficult to tell if resolvers that behave this way are problematic or not, as this could either be a resolver retrying a query over and over rather than falling back to TCP, or a non-caching resolver. Of the observed resolvers 9.7% behaved in this manner.

Van Den Broek et al. propose two solutions to combat fragmentation:

i) Enable the name server to specify the maximum size of a response that they believe to be safe and then take the minimum between that value and the advertised maximum response size. This allows for a simple tweakable parameter to be set, with the potential of no work being required for the operators of the problematic resolvers. However, that can also be seen as a negative as there is now no incentive for the problematic resolver's operator to fix the actual underlying issue. It was found that using 1232 bytes as the parameter had positive results when tested on the SURFnet name server that the observations were made. In the 6.5 hours of testing they preformed, they observed no ICMP FRTE messages, but did notice a very slight increase in truncated UDP responses.

ii) The authors constructed a tool they dub 'DNSRM' which, when combined with a sensor that detects what class of problematic resolver a resolver is, dynamically tunes responses in the attempt to work around the issues the resolver is having. 'DNSRM' acts as a proxy between the resolver and the DNS software running on the name server. This is quite an expensive and complicated mechanism as the sensor requires significant resources in order to do its job. It was found that this system's benefits did not outweigh its complexity versus the first solution discussed.

Fragmenting messages at the Application Layer

Attempts at moving message fragmentation from the transport layer to the application layer in DNS have been proposed and ultimately discarded.

DNS message fragments An IETF draft by Sivaraman et al. [59] was published in 2015 attempting to tackle the problem of fragmenting large DNS message over UDP. The idea is that when a large DNS message cannot fit inside a single UDP packet, split the message up into several new DNS messages which contains as many resource records as they can fit. EDNS0 options are used to signify the number of fragments, the current fragment's

id and if fragmentation is supported by the client. For compatibility reasons, each DNS message that is a fragment of the larger original DNS message should have the truncated flag (TC) set so that a requester can gracefully fall back to TCP. The draft requires that DNS cookies be used in order to avoid DNS amplification and reflection attacks. This draft eventually expired in 2016 until a suspiciously similar draft was published on April 19 2022 with seemingly the only major change being the entire author list.[5] [70]

Additional Truncated Response Originally proposed in 2018, Additional Truncated Response (ATR) [60] is a lightweight way to handle UDP fragmentation for large DNSMessages. The idea is similar to Silvaraman's draft [59], but does not require EDNS options. When a large DNS message is to be sent, simply break it up into several UDP packets, send the first one and wait for some timeout, then send the second one and wait for a timeout, and repeat until all of the UDP packets are sent. This is nice because it does not require a name server to maintain state, and resolvers can implement this behaviour as a daemon rather than having to update the DNS software directly. If a resolver does not support ATR it will simply drop any trailing fragments and fallback to TCP. However, several concerns were raised which were ultimately the demise of ATR. First, some DNS software closes their sockets immediately after receiving a DNS message. This would mean that any trailing fragments would not be deliverable and would cause an ICMP "destination unreachable" packet to be sent back to the name server. This would cause a lot of ICMP noise and would greatly increase the difficulty of debugging networking issues. ATR also produces two times the number of packets for a legitimate exchange, and fives times the number of packets for a malicious one and thus make Denial of Service attacks a concern. Ultimately, combined with the general mindset of "just do not send large DNS messages" of the working group, meant that the ATR draft was rejected.

2.4.3 DNSSEC Configuration

DNSSEC is quite a complex mechanism to deploy, manage, and configure. It requires name servers to pick secure key pairs to use for their signatures, provide those needed keys to the parent zone, and update the parent name servers with said keys when they get changed. In 2016, Dai et al. [19] sought to measure how many active DNSSEC domains, both TLD and SLD, were unable to establish a chain of trust despite indicating they support DNSSEC. Using data provided by ICANN and alexa.com, they found that 0.89% of TLDs and 19.46%

[5]The DNSOP email thread on this topic can be found here:
https://www.mail-archive.com/dnsop@ietf.org/msg25080.html

of SLDs that had DNSSEC deployed were unable to establish a chain of trust. These can be caused by:

i) The registrar not supporting DNSSEC, so the key signing public key is missing from the parent zone, despite the owner configuring their name server to have the extra information needed by DNSSEC.

ii) The parent zone having an outdated key signing public key, which can occur when a rollover takes place.

Dai et al. also observed that RSA was the most common algorithm deployed and that 34% of TLDs and 52% of SLDs that use DNSSEC were using zone signing keys that are too short.[6] Wander [67] also observed these key lengths during the period of April 2013 and January 2017. Part of the thought behind why these key sizes are being used is to reduce potential fragmentation (see Section 2.4.2). Wander recommends exploring using elliptic curve cryptosystems instead of RSA as they provide smaller signatures with signing at the cost of verifying steps being more taxing on a CPU. Dai et al. also found that 16 domains were using even RSA moduli.

2.4.4 DNSSEC Adoption

It may seem odd to discuss DNSSEC adoption in the challenges section, but it is a well-known fact that DNSSEC adoption has been slower than was initially hoped. Wander [67] observed the adoption of DNSSEC between 2013 and 2017. Wander observed that the number of TLDs that supported DNSSEC increased significantly over his observation; starting with just 33% in April 2013 and increasing to 90% in January 2017. Wander notes that this increase is likely due to a massive amount of new *generic* TLDs being introduced during this time. Of the TLDs first observed in April 2013, 52% had deployed DNSSEC by January 2017.

Wander also looked at SLDs by using NSEC and NSEC3[7] to determine which SLDs support DNSSEC. As of January 2017, he found that the SLDs with the highest adoption, as well as the highest number of SLDs using DNSSEC, were .nl, .se, .cz domains all with adoption of over 45%. However, looking at .com shows that having a high number of domains using DNSSEC does not equate to having high adoption. With over six hundred

[6] These RSA keys were less than 1024 bits, with some even being as small as 512 bits.

[7] GPU based dictionary attacks were used to crack the NSEC3 hashes.

thousand .com domains using DNSSEC, that equates to less than 1% of .com domains having DNSSEC enabled.

Wander notes in order to encourage DNSSEC adaoption, .nl provided an 8% discount in registry fees for two years, .cz offered tech support and financial support for marketing, and the Swedish government subsidized discounts for .se domains. He then argues that these national specific domain name initiatives were part of the reason .com saw a jump in DNSSEC signed SLDs. If a registrar already did the work for the national TLDs, then it is an easy thing to deploy to .com domains.

Amplification attacks, UDP/IP fragmentation, configuration, and adoption are all considerable challenges facing DNSSEC, however there are also challenges and obstacles for adding a new signing algorithm to the protocol.

2.5 DNSSEC Algorithm Life Cycle

DNSSEC allows for multiple algorithms to be used for signing responses to queries. So understanding the process an algorithm goes through in order to be widely used is important. Müller et al. [53] analyzed this process for several algorithms.

Standardization takes at least one year and involves an IETF working group to discuss a proposed draft, made by one of the working group members, to determine if they should proceed with working on said draft. Once the working group has settled on a draft a last call is made for feedback before formally asking for the draft to be published. After being reviewed and a wider scoped last call is made, the draft is reviewed by an RFC editor and is then finally published as an RFC.

Müller et al. note that there are several obstacles that algorithms come against when trying to become standardized. Algorithms that are wishing to be standardized must be an improvement to those that are already standardized. Systems that are equally as good are not deemed to be worth the time to standardize as they do not add anything extra to DNSSEC. Software support is also an important requirement: if an algorithm is in a major cryptographic library already then there is less of an obstacle for operators to deploy the algorithm, and thus it is more likely said algorithm will be standardized. Timing and other non-algorithm related issues can also prevent an algorithm from being standardized. Discussions on other key areas may end up preventing progress on standardization, and it can even be deemed that it is the wrong time to add additional algorithm support to DNSSEC.

In order for an algorithm to see usage after being standardized, there needs to be incentive for a zone operator to add support for said algorithm. Cryptography is complex and easy to misconfigure (see Section 2.4.3), so operators are going to avoid potentially breaking something that is currently working just to add something new. Incentives could include financial incentives, such as those mentioned in Section 2.4.4, or an attack on an algorithm they use is released.

In addition to transition between algorithms, entities within the DNSSEC ecosystem must also be able to rollover from one public key to another.

2.6 Root zone key rollover

Refreshing private/public key pairs is important to maintian security. Cryptographic schemes rely on the computational hardness of problems to remain secure; given enough time, any cryptosystem can be broken. Several agencies have recommendations for how long a key of a certain length should be used for.[8] Key rollovers effectively reset attacks being performed against a system using digital signatures, and should be done within the suggested amount of time.

Wander observed [67] that 88% of TLDs rolled over their zone signing keys every 30–120 days and 55% rolled over their key signing key at least once during that four year observation period. The trickiest rollover to perform, however, is the root zone's key signing key.

Recall that the root key signing key is a trust anchor. That is to say, it must be distributed to any system that uses it in a secure manner. This key is used as part of every chain of trust that is constructed on the public internet, so if something goes wrong there is potential for substantial outages. So far, root's key has only been rolled over once, in 2018. Müller et al. [52] reviewed and described the process and determined that all things considered, the rollover was a success.

There are two approaches to distributing the new key to resolvers, *in-band* and *out-of-band*. The out-of-band mechanism is fairly simple but requires additional work by resolver operators. IANA will publish the new key to their website as an XML file, and the resolver operator will download said XML file via some integrity preserving mechanism such as TLS or a digital signature.

[8] keylength.com is a good resource for seeing these recommendations in one place.

In-band is a much more complex system to define, but it removes the work needed to be performed by a resolver operator. First, the new key is added to the DNSKEY set. As resolvers query root for its keys, they will begin to see the new key signing key. Once they start to see this new key, a 30 day *hold-down* period begins. If the new key is seen frequently during this period, the resolver will add that key as a valid public key for the zone that is being queried, in our case root. The reason for this hold-down period is to ensure that any malicious actors who have compromised a trust anchor cannot add their key. Sometime later, the resolver doing the trust anchor rollover will mark the old key as *revoked*, and the key stays in the DNSKEY set. Another 30 day hold-down period commences and concludes with resolvers switching to using the new key and removing the old key from their known keys. Eventually, the old key was removed entirely from the zone's DNSKEY set. The in-band method is supported by several major resolver software packages such as BIND, Unbound, and Knot.

The entire rollover process was quite lengthy and ended up taking two and a half years or five years if initial planning was included. The timeline of deployment was as follows: The replacement key signing key pair was generated in October 2016. The new key was published by IANA in February 2017 and the first signed DNSKEY set including the new key appeared in April 2017. The new key was added to root's DNSKEY set in July 2017, and resolvers began the hold-down process described above. ICANN then paused the rollover process in September 2017 and did not resume the rollover process until a year later. The reason for this delay was that 8% of resolvers that operators were observing were signalling that they did not have the new key.

It was only one month after the rollover process resumed that the time to live for the old key expired, and the old key was marked as revoked in January 2019. Finally, the old key was removed from the root zone DNSKEY set in March 2019.

Müller et al. observed that when there were configuration problems that caused outages, a majority of operators had fixed the outages within an hour. They did note however that telemetry used for these observations, both for operators and themselves, was less than ideal. Although RFC 8145 [69] introduces a way for zone operators to test key deployment and saw steady adoption prior to the rollover, one could not retrieve stateful information from a resolver due to it being passively collected. The good news is RFC 8509 [36], root sentinel, solves that particular issue and is seeing wide adoption. Müller et al. note that both solutions suffer from potential signal distortion and lack of query volume metrics.

Müller et al. also observed an interesting trend of the number of resolvers indicating they supported the old key increased after the key was revoked. This is likely due to older versions of software which included the old key being used.

Chapter 3

Post-quantum Cryptography

Public-key cryptography bases its security on the difficulty of solving certain problems. If one of these problems were discovered to be easy to solve, then all cryptoschemes that are built off of that problem would also be easy to break and therefore not secure.

Quantum computers unlock a new way to solve problems that are considered hard using a classical computer. Classical computers use bits to represent data and can store either a 0 or a 1. Quantum computers, on the other hand, represent data using quantum bits, or qubits, and take advantage of multiple properties of quantum mechanics; namely quantum superposition, quantum entanglement, and quantum inference.

In 1994 an efficient factoring quantum algorithm was unveiled by Shor [58]. Once a sufficiently powerful quantum computer is developed, Shor's algorithm can be used to efficiently break the widely deployed public-key cryptoschemes we use today. Major entities such as Google, IBM, and Microsoft are all researching and developing quantum computing, and in some cases, are even offering cloud based quantum computing products [30, 37, 47].

Although quantum computers are not an active threat to our deployed cryptography today, it is important to proactively prepare for the day that is no longer the case.

3.1 NIST Post-Quantum Cryptography Selection Process

NIST is responsible for releasing Federal Information Processing Standards (FIPS) which are used as guidelines by government agencies when there is not already a pre-existing

industry solution. Although NIST releases FIPS for government agencies, many in the private sector also follow these standards [55]. NIST has standardized several cryptographic primitives in the past such as AES [23], SHA-1 [42], SHA-2 [20], and SHA-3 [22]. Not all of these were standardized in the same way. Some, such as SHA-1 and SHA-2 were developed internally by the National Security Agency (NSA), whereas others took a more open approach and took submissions and comments from the global cryptographic community before ultimately settling on the algorithm to standardize for government use. This open process brings more expert scrutiny to the selection process thus decreasing the likelihood of security flaws being discovered post standardization.

Looking to the future, NIST posted a Call for Proposals for its post-quantum cryptography standardization process on December 16 2016 [54]. The goal for this open selection process is to standardize a set of post-quantum public-key encryption/key encapsulation mechanisms and a set of post-quantum digital signature algorithms so that the transition can begin well before a realistic quantum adversary is developed. NIST defined five levels of security which algorithms should aim to achieve with level 1 being the least secure and level 5 being the most secure. The definitions of the NIST security levels are in Table 3.1.

Table 3.1: NIST Security levels

Level	Security Requirement
1	At least as secure as AES128
2	At least as secure as SHA256/SHAKE256
3	At least as secure as AES192
4	At least as secure as SHA384/SHAKE384
5	At least as secure as AES256

On December 21, 2017 the process began with a pool of 82 digital signature and public-key encryption/key encapsulation mechanisms with 13 not meeting the minimum acceptance criteria, and 5 being withdrawn [2]. After a round of public comments, the set of algorithms was narrowed to 26 potential candidates on January 30, 2019 [50]. After another round of public comments, the pool of algorithms was narrowed to six (three finalists, three alternates) digital signature algorithms and nine (four finalists, fives alternates) public-key encryption/key encapsulation mechanisms. On July 5 2022, NIST announced the key encapsulation algorithm CRYSTALS-KYBER [9] was selected for standardization, and that CRYSTALS-Dilithium [10], Falcon [28], and SPHINCS+ [8] were selected as the digital signature algorithms to be standardized [3]. A fourth round has been announced and exclusively includes public-key encryption/key encapsulation mechanisms. Round four algorithms consist of Binary Flipping Key Exchange (BIKE) [5], Classic McEliece [4], Ham-

ming Quasi-Cyclic (HQC) [1], and Supersingular Isogeny Key Encapsulation (SIKE) [39]. A Call for Proposals for additional digital signature schemes was published on September 6, 2022 with a submission deadline of June 1, 2023 [56].

3.2 Selected Algorithms and Round 4 Candidates

In this section we summarize both the selected algorithms (denoted with †) and the current round four candidates (denoted with ‡) of the NIST post-quantum cryptography selection process.

3.2.1 Digital signature algorithms

CRYSTALS-Dilithium† CRYSTALS-Dilithium's security is based on the hardness of Module Learning With Error problem and is available in two different flavours. Standard Dilithium uses SHAKE-128 to expand the matrix in the public and private keys, whereas Dilithium-AES uses, unsurprisingly, AES for matrix expansion. Dilithium provides parameter sets which achieve NIST security levels 2, 3 and 5. CRYSTALS-Dilithium was selected as the primary signature algorithm because of its high efficiency, simple implementation, strong theoretical security basis, and its positive cryptanalysis history. Table 3.2 depicts Dilithium's submitted parameter set's security level and public key, secret key and signature sizes.

Table 3.2: CRYSTALS-Dilithium's parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
Dilithium2	2	1312	2528	2420
Dilithium3	3	1952	4000	3293
Dilithium5	5	2592	4864	4595

Falcon† Falcon's security is based on the hardness assumption of the shortest integer solution over NTRU lattice problem. The Falcon team submitted two parameter sets, Falcon-512 and Falcon-1024 which achieve NIST security levels one and five respectively. Falcon supports several types of signature formats: *compressed*, *padded*, and *CT*. *Compressed* signatures are variable in length and are the smallest signatures on average. *Padded* signatures are fixed length. *CT* signatures are fixed length and allow for constant-time processing in the context of message data and signature value. Falcon was selected for standardization due to its strong security and its reasonably sized signatures and public keys. It is noted that the algorithm is quite complex and could therefore lead to implementation bugs compromising security in practice. Table 3.3 depicts Falcon's submitted parameter set's security level and public key, secret key and signature sizes. Recent work has been able to shrink Flacon signatures by up to 40% with a minimal security cost [27], however as this work has yet to be added to the Falcon specification we exclude the results from Table 3.3

Table 3.3: Falcon's parameter sets

* denotes the average size over 10,000 signatures (100 secret keys, 100 signatures per key) plus/minus the standard deviation.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
Falcon-512 compressed	1	897	1281	651.59±2.55*
Falcon-512 padded	1	897	1281	666
Falcon-512 CT	1	897	1281	809
Falcon-1024 compressed	5	1793	2305	1261.06±3.57*
Falcon-1024 padded	5	1793	2305	1280
Falcon-1024 CT	5	1793	2305	1577

SPHINCS+† SPHINCS+ is a hash-based signature scheme and has three instantiations which use the hashing algorithms SHA2, SHAKE, and HARAKA. For each instantiation, a "small" and a "fast" set of parameters are provided. SPHINCS+ imposes a maximum number of signatures that should be generated for a particular public key. For each signature created, the initial low probability that enough of the private key will be revealed to the attacker increases. NIST required in its original call for proposals that 2^{64} signatures must be safely generated. SPHINCS+ is quite complex which could lead to implementation issues. SPHINCS+ has similar assumptions to NIST's already standardized stateful hash-based signature algorithms; however SPHINCS+ does not require the signer to maintain state. SPHINCS+ signatures are also extremely large, with the smallest of the proposed

parameter sets being over 7 kilobytes. There are parameter sets which achieve NIST security levels one, three, and five. SPHINCS+ was selected for standardization due to its strong security and because SPHINCS+ relies on a different security assumption compared to CRYSTALS-Dilithium and Falcon which both use lattices. SPHINCS+ therefore provides a strong alternative in the event that attacks are found that break the lattice based algorithms. Table 3.4 depicts SPHINCS+'s submitted parameter set's security level and public key, secret key and signature sizes.

Table 3.4: SPHINCS+'s parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
SPHINCS+-128s	1	32	64	7,856
SPHINCS+-128f	1	32	64	17,088
SPHINCS+-192s	3	48	96	16,224
SPHINCS+-192f	3	48	96	35,664
SPHINCS+-256s	5	64	128	29,792
SPHINCS+-256f	5	64	128	49.856

3.2.2 Public-key encryption/key encapsulation mechanisms

Although this thesis foscuses on applying post-quantum digital signatures to DNSSEC, we include the public-key encryption/key encapsulation mechanisms selected for the fourth round of NIST's standardization processes for completeness.

CRYSTALS-KYBER[†] CRYSTALS-KYBER's security is based on the Module Learning With Errors problem. CRYSTALS-KYBER has two versions; the standard version which uses SHAKE, and the "90s" version which uses a combination of AES in counter mode and SHA2. CRYSTALS-KYBER has suggested parameter sets which achieve NIST security levels of one, three, and five. CRYSTALS-KYBER was selected because of its thorough security analysis and its strong performance. Table 3.5 depicts CRYSTALS-KYBER's submitted parameter set's security level and public key, secret key, ciphertext, and shared secret sizes.

BIKE[‡] BIKE is based on the Quasi-Cyclic Moderate Syndrome Decoding and Quasi-Cyclic Codeword Finding problems and provides parameter sets that achieve NIST security

Table 3.5: CRYSTALS-KYBER's parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Ciphertext Size (bytes)	Shared Secret Size (bytes)
KYBER512	1	800	1,632	768	32
KYBER768	3	32	64	17,088	32
KYBER1024	5	48	96	16,224	32

levels one and three. NIST selected BIKE as a round four candidate due to its strong performance. Table 3.6 depicts BIKE's submitted parameter set's security level and public key, secret key, ciphertext, and shared secret sizes.

Table 3.6: BIKE's parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Ciphertext Size (bytes)	Shared Secret Size (bytes)
BIKE-L1	1	1,541	5,223	1,573	32
BIKE-L3	3	3,083	10,105	3,115	32

Classic McEliece[‡] Classic McEliece is built on Neiderreiter's dual version of McEliece's public key encryption using binary Goppa codes. There are two versions of Classic McEliece: a simpler "systematic" version and a generalized "semi-sytematic" version. Classic McEliece submitted parameter sets provide NIST security levels of one, three, and five. NIST selected Classic McEliece as a round 4 candidate as they are confident in its security, however do not see use cases for the scheme do to its extremely large public key size. NIST are using round 4 to determine if there is an application for Classic McEliece, and leaving Classic McEliece as an option in the event the other round 4 candidates are determined to be insecure. Table 3.7 depicts Classic McEliece's submitted parameter set's security level and public key, secret key, ciphertext, and shared secret sizes.

Table 3.7: Classic McEliece's parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Ciphertext Size (bytes)	Shared Secret Size (bytes)
Classic McEliece 348864	1	261,120	6,452	128	32
Classic McEliece 460896	3	524,160	13,568	188	32
Classic McEliece 6688128	5	1,044,922	13,892	240	32
Classic McEliece 6960119	5	1,047,319	13,908	226	32
Classic McEliece 8192128	5	1,357,824	14,080	240	32

HQC‡ HQC is based on the Quasi-cyclic Syndrome Decoding with parity problem and provides parameter sets which achieve NIST security levels of one, three, and five. NIST selected HQC as a round 4 candidate due to its strong security, however they are concerned by both HQC's public key and ciphertext sizes. Table 3.8 depicts HQC's submitted parameter set's security level and public key, secret key, ciphertext, and shared secret sizes.

Table 3.8: HQC's parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Ciphertext Size (bytes)	Shared Secret Size (bytes)
HQC-128	1	2,249	2,289	4,481	64
HQC-192	3	4,522	4,562	9,026	64
HQC-256	5	7,245	7,285	14,469	64

SIKE‡ SIKE is based on the Supersingular Isogeny Walk problem, and provides parameter sets which achieve NIST security levels of one, two, three, and five. SIKE offers a default version as well as a compressed version which trades smaller public keys and ciphertexts in exchange of performance. SIKE was selected as a round 4 candidate due to its small key sizes, but due to SIKE being relatively new it requires additional scrutiny. On July 30, 2022, a devastating attack [15] on SIKE was published removing SIKE from contention, but we include its parameter sets for completeness. Table 3.9 depicts SIKE's submitted parameter set's security level and public key, secret key, ciphertext, and shared secret sizes.

Table 3.9: SIKE's parameter sets.

Parameter Set	Security Level	Public Key Size (bytes)	Secret Key Size (bytes)	Ciphertext Size (bytes)	Shared Secret Size (bytes)
SIKE-p434	1	330	374	364	16
SIKE-p434 compressed	1	197	350	236	16
SIKE-p503	2	378	434	402	24
SIKE-p503 compressed	2	225	407	280	24
SIKE-p610	3	462	524	486	24
SIKE-p610 compressed	3	274	491	336	24
SIKE-p751	5	564	664	596	32
SIKE-p751 compressed	5	335	602	410	32

3.3 Stateful Hash-Based Signatures

The NIST Post-quantum Cryptography standardization process is not the first time NIST has standardized algorithms which are resistant to a quantum adversary. Prompted by the IETF standardizing the stateful hash-based signature schemes XMSS with RFC 8391 [34] and LMS with RFC 8554 [46], NIST requested public input on whether they should standardize the two schemes. Stateful Hash-Based signatures are different from the digital signature schemes described above as the signer must maintain a state and that a signing key has a limited number of signatures it can produce. Maintaining state can be problematic in certain scenarios such as distributed systems, however stateful hash-bashed signing algorithms often have smaller keys and are more performant. On October 30 2020 NIST published its recommendation for XMSS and LMS. It is worth noting that NIST only recommends a subset of the parameter sets defined in RFC 8361 and RFC 8554 [18].

3.4 Post-Quantum Cryptography and the Domain Name System

DNSSEC relies on classical public-key cryptography to maintain message integrity. Once a powerful enough quantum computer is developed, the currently standardized algorithms will leave DNSSEC vulnerable and untrustworthy. As part of NIST's call for proposals [54], DNSSEC was listed as one of the protocols which needs to be hardened against a quantum adversary. As both IETF standardization and root key rollovers can take years, it is important to start the discussion and planning now, so that we can be prepared for the

inevitable quantum future.

Müller et al. [51] began this discussion by evaluating the round 3 NIST candidates in the context of DNSSEC. They established several requirements for a scheme to fulfil if it were to be used for DNSSEC signatures. As discussed in Section 2.4.2, fragmentation is a major concern for DNSSEC and the recommended maximum response size, both for signatures and public keys, should not exceed 1232 bytes. However, due to public keys not needing to be transmitted as often as signatures, larger public keys may be acceptable. They also require that a resolver be able to validate 1000 signatures per second as currently a "medium" sized resolver only performs on the order of hundreds of validations a second.[1] In order to support large, frequently changing zones, the final requirement is that the algorithm must be able to create 100 signatures a second.

Three of the round 3 candidate algorithms fulfil the requirements sufficiently enough to be considered: Falcon-512, Rainbow-I_a [21] and RedGeMSS128 [14].[2] On first inspection it would appear that Falcon-512 is the clear winner as it is the only scheme that completely meets the requirements set above, in particular its signature and public key sizes both being within the 1232 byte limit. However, both Rainbow-I_a and RedGeMSS128 have significantly smaller signatures sizes which makes them extremely appealing. Falcon-512 has a signature size of 0.7kB whereas the other two schemes have signature sizes of 66 bytes and 35 bytes respectively. The requirement that both Rainbow-I_a and RedGeMSS128 fail is their public keys are 158kB and 375kB respectively vs Falcon-512's comparatively paltry key size of 0.9kB. Falcon-512 may not be the drop in replacement we are looking for, however, as only a single key or signature can fit within the 1232 byte threshold. Müller et al. expect that DNSSEC specification changes will be required before quantum safe cryptography can be deployed in order to support larger key and/or signature sizes.

Müller et al. discuss two potential changes that could solve the large key and signature size issue described above, the first being TCP. Currently, DNS resolvers can revert to TCP when they receive a truncated response from a zone, however 11% of name servers do not support TCP which makes this option not viable. The good news is DNS Flag Day 2020 is championed TCP, and support is improving, however more work is required [45]. The second option is to use encrypted DNS such as DNS-over-TLS or DNS-over-HTTPS which both rely on TCP and are both "gaining traction". Ultimately, Muller et al. state that changes to the DNSSEC protocol will likely be required.

[1] As DNSSEC has not yet hit widespread adoption it is likely that this will grow, however it is likely that cryptography libraries will also improve their efficiency as well.

[2] Note: Rainbow-I_a and RedGeMSS128 did not advance into round 4 of NIST's selection process as both had major attacks compromising their security discovered during round 3.

One proposed modification that Beernink presented in their thesis [11] was delivering DNSKEY delivery out-of band rather than in standard DNS messages. The idea is that when a large DNSKEY is required, such as when using the now defunct round 3 candidate Rainbow [21], for verification the requesting server would initiate a HTTP or FTP request to fetch the large key. Unforunately none of the algorithms selected by NIST provide the large DNSKEY small signature trade-off that Beernink was designing for, and thus other options must be explored.

Chapter 4

Adapting DNSSEC for a Quantum Future

As the threat of a powerful quantum adversary looms, we must begin the process of planning for, and adopting, quantum secure algorithms in mission critical pieces of infastructure, such as DNSSEC. The three digital signing algorithms selected by NIST all have substanially larger signatures than what are currently standardized for use with DNSSEC causing larger DNS messages to be sent. In this chapter we intoduce A Resource Record Fragmentation mechanism as a request based fragmentation mechanism for handling the larger DNS messages that the NIST selected algorithms necessitate. We construct a simple lightweight proof of concept daemon which implements ARRF and only requires minor modifications to DNS software. Specifically, we increase several internal buffers to 65535 bytes to account for the largest possible DNS message size. We then evaluate DNSSEC resolution times using the three NIST selected post-quantum digital signature algorithms both with and without the ARRF daemon.

4.1 Request based fragmentation

As discussed in Section 3.4 changes will need to be made in order to support post-quantum cryptography in DNSSEC. In a perfect world, we could simply send the larger DNS messages with little to no concern of them arriving. However, as described in Section 2.4.2, UDP/IP fragmentation can cause significant problems for delivering large DNS messages via UDP. The current solution to solving this problem is falling back to TCP, however,

a non-trivial amount of DNS name servers do not support TCP. We look to solve this problem by moving DNS message fragmentation from UDP (transport layer) to DNS itself (application layer), while addressing concerns raised to previously proposed mechanisms. We dubbed our solution A Resource Record Fragmentation mechanism, or ARRF for short.

4.1.1 Resource Record Fragments

When a DNS message is too large to fit into the maximum advertised UDP size, the message must be shrunk while still containing meaningful information to the requester. We introduce a new type of pseudo-resource record: Resource Record Fragments (RRFrags). Like OPT, another pseudo-resource record, RRFrags are not explicitly in DNS zones. Rather they are created only when they are needed. RRFrags are designed similarly to the OPT pseudo-resource record; they use the standard resource record wire format but repurpose some of the fields. An RRFrag contains the following fields:

NAME Must always be root (.) to reduce the amount of overhead required to send a RRFrag while respecting the generic resource record format.

TYPE Used to identify that this pseudo-resource record is an RRFrag.

RRID This is used to indicate the particular resource record that is being fragmented. Since labels do not necessarily have distinct resource records attached to them, this allows a requester to be explicit in its request while not requiring the responder to remember which particular resource record it fragmented. The RRID of a particular resource record can be arbitrarily assigned, but must not change.

CURIDX This is the current index in the byte array of the original resource record which is being fragmented.

FRAGSIZE This is the total number of bytes contained in FRAGDATA plus two bytes to account for the extra space needed for the RRSIZE field. FRAGSIZE has two different meanings depending on the context. If the RRFrag is part of a query, then this indicates how large the responding server should make this particular fragment. If the RRFrag is part of a response, this field indicates how much data was sent in this particular fragment.

RRSIZE This is the size of the original non-fragmented Resource Record. This is used by the requester to determine how much data it still needs to request from the responder in order to reassemble that particular resource record.

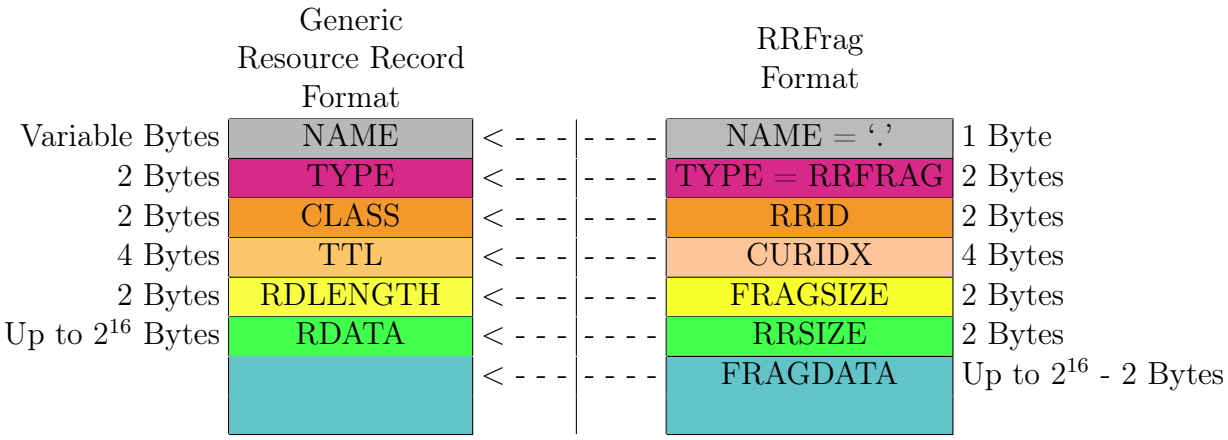


Figure 4.1: The mapping of the RRFrag format onto the generic resource record format.

FRAGDATA This field contains the raw bytes of the fragment. In queries this is always empty. In responses this will contain FRAGSIZE bytes starting at CURIDX. It is possible for FRAGDATA to contain zero bytes in responses, which we will elaborate on later.

Figure 4.1 depicts how an RRFrag maps onto the generic resource record format. Note that similar to a DNSKEY resource record where the extra fields needed required are inside RDATA, an RRFrag stores the RRSIZE alongside FRAGDATA. This was done to handle the case where an implementation which does not support ARRF blindly copies RDLENGTH, or in our case, FRAGSIZE bytes into a buffer prior to branching based on resource record type.

4.1.2 Using RRFrags

When a DNS response is too large to fit in the maximum advertised UDP size, RRFrags are used to shrink the response until it is below the advertised threshold. Resource records are replaced with RRFrags in place. That is to say, if a resource record being fragmented is in a particular section of the DNS message, the RRFrag replacing will be inserted into the same section. This is key so that the original message format once all resource records are assembled will remain intact. It is important to note that the OPT pseudo-resource record must not be fragmented as it contains important meta data about the response, such as the DNS cookie. DNS messages that contain RRFrags should send as much data as they are able without surpassing the advertised threshold.

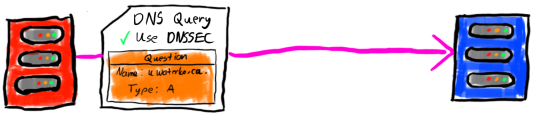
The initial response containing at least one RRFrag can be considered a "map" of the non-fragmented message. This map is used by the requester to determine what the non-fragmented DNS message will look like upon reassembly. The requester can now determine what fragments it is missing in order to complete the original large DNS message, and can now send a new query for the missing RRFrags. It is the responsibility of the requester to specify which resource records it desires, how large the fragments should be, and where the fragments start. This is done by adding a RRFrag for each distinct RRID the requester is requesting a fragment for in the query's additional section. If the response contains any non-RRFrag resource records, it should store them until it is possible to reassemble the entire DNS message.

When the responder sees a query containing a RRFrag, it just has to construct a standard DNS response by inserting the corresponding RRFrags into the answers section. The Fragdata being sent is a simple copy of the bytes of the desired resource record starting at CURIDX and ending at CURIDX + FRAGSIZE. This request response cycle continues until the requester is able to reassemble the original large non-fragmented message. For backwards compatibility reasons, whenever a response is sent which contains an RRFrag, the truncated flag (TC) must be set in the DNS message header. Setting the truncated flag also removes the need to advertise support for ARRF as a resolver that does not support ARRF will retry the request over TCP.

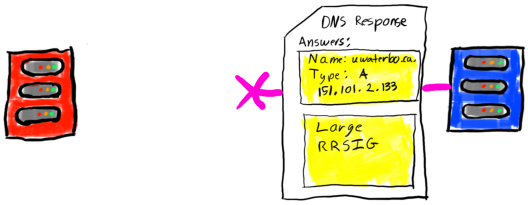
If a requester asks for a fragment which cannot be constructed, such as an RRID which does not map to a specific resource record, the responder should respond with a return code of FORMERR to indicate that the query was malformed.

4.1.3 Example execution of ARRF

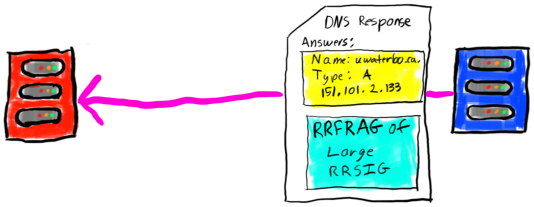
To better solidify how ARRF works, we will now work through an example DNS query whose response is larger than the MTU. This example has had some details abstracted away and should not be used in place of the above specification when implementing ARRF. Figure 4.2 illustrates our example execution. This example begins at the last stage of name resolution for the query "uwaterloo.ca.". We have two servers; the resolver (coloured red) making the DNSSEC enabled query for uwaterloo.ca., and the uwaterloo.ca. name server (coloured blue).



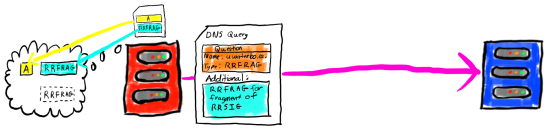
(a) Resolver sends a DNSSEC enabled request for uwaterloo.ca.'s A record



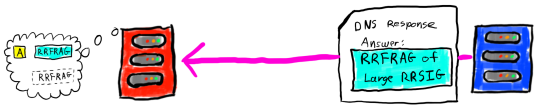
(b) The name server constructs a response which is too large to send within the MTU limit



(c) The name server replaces the large RRSIG with an RRFRAG containing as much of the RRSIG as possible

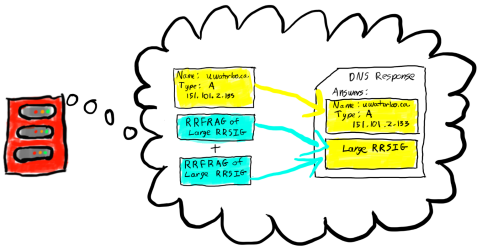


(d) The resolver receives the DNS response, copies the A record and RRFRAG data into its state and sends a request for the remaining fragment of the large RRSIG



(e) The name server constructs a DNS response containing the requested fragment and sends the response to the resolver

(f) The resolver copies the missing fragment into its state.



(g) The resolver combines the two RRFrags it received and reconstructs the large original DNS response

Figure 4.2: A simplified example execution of ARRF

First the resolver makes a standard request for the A record and its associated RRSIG (Figure 4.2(a)). Upon receiving the request, the resolver observes that the DNS response is too large to fit within the confines of the MTU (Figure 4.2(b)), and thus replaces the large RRSIG with an RRFrag (Figure 4.2(c). This RRFrag will contain as much of the original RRSIG as possible, and will inform the resolver how much of the original RRSIG is missing. Once the resolver receives the DNS response, it will copy both the entire A record as well as the RRFrag (Figure 4.2(d)) and allocating enough space for the rest of the missing record. The resolver will then send another DNS query, but this time asking for an RRFrag and sending its own RRFrag (Figure 4.2(e)). Once the uwaterloo.ca name server receives the RRFrag query, it will use the RRFrag in the additional section to determine the starting position and size of the fragment of the original RRSIG is being requested. The name server will construct a new DNS response containing the rest of our missing RRSIG inside of an RRFrag and send the new response to the resolver (Figure 4.2(f)). Finally the resolver will copy the newly received RRFrag into its state, reassemble the original RRSIG, and reconstruct the original large DNS response (Figure 4.2(g)). DNSSEC validation now takes place, and if verification is successful the records are cached by the resolver.

4.1.4 Caching and DNSSEC Considerations

RRFrags themselves should never be cached. Once a DNS message is reassembled, and if necessary DNSSEC validated, then non-fragmented resource records may be cached. If RRFrags could be cached, this would allow for malicious data to be accepted prior to validation. Caching complete resource records as opposed to RRFrags also allows for intermediate resolvers to send different fragment sizes then they originally received which allows for more flexibility to handle varying advertised UDP sizes. For resolvers which do not support ARRF RRFrags will not be cached as they are sent in truncated messages resulting in the resolver falling back to TCP.

4.2 Evaluating Post-quantum DNSSEC

In this section we evaluate the three algorithms selected by NIST at the end of round 3 of their Post-quantum selection, Falcon, Dilithium, and SPHINCS+, as well as include results for RSA 2048 with SHA256 and ECDSA P256 for the sake of comparison. We evaluate these algorithms both using DNSSEC as defined today, as well as with ARRF. To perform this evaluation we used Internet Systems Consortium's BIND9 9.17.9 [38] as our DNS server software. We then added support for the three selected algorithms to BIND9

using Open Quantum Safe's liboqs 0.7.1 and OpenSSL 1.1.1l fork [61]. To deploy a test DNS network we used Docker and Docker's built in networking as well as 'tc' to simulate network bandwidth and latency. Finally, rather than implementing ARRF directly into BIND9, we constructed a daemon which intercepts all incoming and outgoing network traffic and implements ARRF transparently for both the resolver and all name servers. We use libnetfilter-queue 1.0.3-1 to intercept packets.

We will now describe how the daemon behaves. When the machine acting as the name server receives a DNS query, it will modify the maximum advertised UDP message size to the maximum value of 65535 bytes[1]. The daemon then sends the message to the DNS software, which will then respond with a UDP message up to 65535 bytes. If ARRF were implemented directly into BIND9, DNS messages could be arbitrary in size as BIND9 could digest resource records as they are received (and validated), rather than needing the entire message to be received. The daemon then receives this response, copies the entire message into its state and effectively becomes the DNS name server. Whenever a fragment is requested in the future the daemon will use its state if possible rather than sending the request to the DNS software. For simplicity, our daemon never flushes its state; however, should ARRF be deployed, state consistency will need to be maintained. When the machine acting as the DNS resolver receives a DNS response containing an RRFrag, the daemon will intercept the message. The daemon will create a state for that individual transaction containing the metadata provided by the initial response's map and copy any data included into the state. The daemon will then execute ARRF until the entire message can be reconstructed, and the daemon transparently sends the reconstructed message to the DNS resolver software.

To evaluate Falcon, Dilithium, and SPHINCS+, we construct a simple DNS network consisting of a client, a resolver, and a name server each running in their own Docker container on the same machine. The name server zone contains 1000 'A' records, each with a unique label and signature. We query for each of these A records and measure the total resolution time for each one. The zone also contains 1 'primer' name record which is used to 'prime' the resolver so that it does not perform and status queries and already has the zone's Zone Signing Key. To model the worst case response size, we disabled 'minimal responses' and as such each response will contain 1 question, 1 A record, 1 NS record, 1 SOA record, and 3 RRSIGs. We use '`dig`' to issue each query and measure the total resolution time of said query.

We evaluate these algorithms using a variety of network conditions. To model a low bandwidth low delay setting, we use a network with 10ms of delay and 128 kilobytes per

[1]Modifications to BIND9 were required as the maximum DNS message size BIND9 supports is 4096

second bandwidth. We also evaluate a low delay high bandwidth network, we use 10ms of delay and 50 megabytes per second bandwith. A high latency moderate bandwidth network is also evaluated with 100ms of delay and 50 megabytes per second bandwidth. Finally, we measure an ideal network where there is 0 delay, and unlimited bandwidth, with the only cost being processing the messages. All experiments were run on a c5.2xlarge Amazon Web Services instance which provides 8 cores of an Intel Xeon(R) Platinum 8124M with 16 gigabytes of RAM.

4.2.1 Algorithm performance

In order to fully understand the results, it is important to understand the performance of the verification function of each of the algorithms. We use the Open Quantum Safe openssl fork's speed command to measure each algorithms signing and verification performance and report the results in Table 4.1.

Table 4.1: Algorithm runtime measured using OpenSSL Speed

Algorithm	Sign (ms)	Verify (ms)
Falcon-512	0.2810	0.0438
Dilithium2	0.0753	0.0268
SPHINCS+-SHA256-128s	373.1	1.360
RSA 2048 with SHA256	0.5600	0.0177
ECDSA P256	0.0219	0.0677

4.2.2 Post-quantum with standard DNSSEC

We first measure how the Post-quantum algorithms perform if they were deployed to DNSSEC as it is currently specified under two scenarios and five different network conditions. We first measure how the algorithms would perform with a maximum UDP size of 1232. For messages larger than 1232 bytes, the DNS servers will fall back and establish a new TCP connection. The second scenario is the exclusive use of UDP without fragmentation for DNS communication provides an idealized view of the best case performance we can achieve using a particular algorithm. The average resolution times with standard deviation for the various network conditions are depicted in Table 4.2. RSA 2048 with SHA256 and ECDSA P256 only have entries for Standard DNS as the signatures of these algorithms are small enough to ensure they can fit in a single DNS message without fragmentation.

Table 4.2: Resolution times plus or minus standard deviation without ARRF

Algorithm	Standard DNS Resolution Time (ms)	DNS using only UDP Resolution Time (ms)
10ms of latency and 128 Kilobytes per second bandwidth		
Falcon-512	107.3 ± 1.8	61.5 ± 2.2
Dilithium2	147.9 ± 1.5	102.0 ± 1.9
SPHINCS+-SHA256-128s	275.4 ± 2.1	229.4 ± 2.0
RSA 2048 with SHA256	52.20 ± 1.2	N/A
ECDSA P256	47.78 ± 1.9	N/A
10ms of latency and 50 Megabytes per second bandwidth		
Falcon-512	82.11 ± 2.3	40.6 ± 2.1
Dilithium2	82.24 ± 2.2	40.8 ± 2.3
SPHINCS+-SHA256-128s	82.59 ± 2.1	41.2 ± 2.2
RSA 2048 with SHA256	41.50 ± 2.2	N/A
ECDSA P256	47.49 ± 1.9	N/A
100ms of latency and 50 Megabytes per second bandwidth		
Falcon-512	802.1 ± 2.1	401.6 ± 2.0
Dilithium2	802.4 ± 2.0	401.5 ± 2.0
SPHINCS+-SHA256-128s	802.5 ± 1.9	401.9 ± 2.0
RSA 2048 with SHA256	401.3 ± 2.0	N/A
ECDSA P256	401.2 ± 2.2	N/A
0ms of latency and unlimited bandwidth		
Falcon-512	2.5 ± 3.9	1.1 ± 2.0
Dilithium2	2.3 ± 3.3	1.2 ± 2.2
SPHINCS+-SHA256-128s	2.4 ± 3.5	1.2 ± 1.9
RSA 2048 with SHA256	1.7 ± 3.0	N/A
ECDSA P256	1.6 ± 2.7	N/A

4.2.3 Post-quantum with ARRF

In this section we evaluate how each of the algorithms perform when using two different flavours of ARRF. First we offer a "sequential" version. This version sends a request, receives a response, then looks what it needs to request and sends another request. This process is repeated until the entire message is received. Secondly we offer a parallel version where once the first response is received the name server sends all of the requests for fragments at once essentially parallelizing the ARRF mechanism. We provide both of these variations as we hypothesize that should ARRF be standardized and deployed that

operators may wish to rate limit ARRF messages to prevent overwhelming middle boxes. These two versions of ARRF provide the two extremes of ARRF's performance. Should rate limiting be used, the observed performance would be between those two extremes. We provide several scenarios where the maximum DNS message size varies across all of the various network conditions described above.

Our daemon implementation is far from production ready, and should be treated as a proof of concept. With that in mind, it is important to understand the raw overhead that the daemon incurs. By setting the maximum DNS message size to 65535 bytes, we will see how much of a cost we are paying just by running the proof of concept daemon. We then evaluate what we would expect most operators would use as their maximum DNS message size of 1232 bytes. In order to see how ARRF scales, we also provide some smaller maximum DNS message sizes of 512[2] and 256 bytes. Table 4.3 depicts the measured mean resolution time in milliseconds for the sequential daemon across the various network conditions measured, and Table 4.4 contains the results for the parallel daemon. Figures 4.3, 4.4, 4.5, and 4.6 illustrate all measured resolution times for standard DNS and DNS using ARRF for all network conditions. Each point is the average resolution time over 1000 DNS lookups for the various algorithms being evaluated both with and without ARRF. When ARRF is used, maximum message sizes of 65535, 1232, 512, 256 are included.

4.2.4 Post-quantum data transmission

In order to understand the full implications of deploying ARRF, we must also consider the amount of data transmitted compared to that of the DNS as it is currently standardized. We measured the total amount of bytes required to transmit a complete DNS message signed with Falcon-512, Dilithium2, and SPHINCS-SHA256-128s both with and without ARRF deployed. The results are located in Table 4.5.

4.2.5 Results

Regardless of network conditions and whether ARRF was used or not, we can see trends across the evaluated algorithms. When considering standard DNS, RSA and ECDSA have the shortest resolution times with the best performing post-quantum algorithm being twice as slow across all network conditions. This is due to the response sizes being too large for a single UDP packet, causing it to be truncated and thus effectively making the initial

[2]The minimum DNS message size that must be supported

Table 4.3: Resolution times plus and minus standard deviation with the sequential ARRF daemon

Algorithms	Resolution Times (ms) for each Maximum Message Size (Bytes)			
	65535	1232	512	256
10ms of latency and 128 Kilobytes per second bandwidth				
Falcon-512	62.6 ± 2.1	84.4 ± 1.5	148.5 ± 1.6	275.8 ± 1.7
Dilithium2	103.2 ± 1.8	231.7 ± 1.8	422.7 ± 2.4	803.9 ± 1.3
SPHINCS+-SHA256-128s	230.7 ± 1.9	635.1 ± 2.1	1271.0 ± 2.0	2480.0 ± 1.9
10ms of latency and 50 Megabytes per second bandwidth				
Falcon-512	41.8 ± 2.1	62.1 ± 2.3	122.5 ± 2.2	243.0 ± 2.3
Dilithium2	41.9 ± 2.1	162.9 ± 2.2	343.8 ± 1.9	705.6 ± 2.4
SPHINCS+-SHA256-128s	42.5 ± 2.2	424.7 ± 1.8	1028.0 ± 2.5	2173.0 ± 2.1
100ms of latency and 50 Megabytes per second bandwidth				
Falcon-512	402.0 ± 2.1	601.1 ± 2.9	1203.0 ± 1.9	2404.0 ± 1.1
Dilithm2	402.1 ± 2.0	1604.0 ± 1.8	3405 ± 2.1	7008.0 ± 1.7
SPHINCS+-SHA256-128s	402.7 ± 2.0	4207.0 ± 2.0	10210 ± 1.8	21620.0 ± 1.4
0ms of latency and unlimited bandwidth				
Falcon-512	1.6 ± 2.3	2.0 ± 2.6	2.2 ± 2.4	2.7 ± 2.6
Dilithium2	1.8 ± 2.6	2.3 ± 2.5	2.9 ± 2.2	4.2 ± 1.3
SPHINCS+-SHA256-128s	2.0 ± 2.4	3.6 ± 1.5	5.7 ± 2.2	5.7 ± 2.4

query a wasted trip. The resolver must then fall back to the less performant TCP protocol to complete the lookup. When only UDP is used, ECDSA and RSA only beat Falcon-512 and Dilithium2 when bandwidth was restricted to 128 kilobytes per second. This is likely due to the verification functions of Falcon-512 and Dilithium2 being more efficient than ECDSA and RSA.

When considering the cases where the ARRF daemon is running, but not actively fragmenting resource records, we see comparable performance to standard DNS using only UDP. When comparing the post-quantum algorithms on standard DNS using only UDP versus the ARRF daemon using a maximum message size of 65535 bytes we see a minimal overhead never exceeding 1.25 ms. As previously mentioned our daemon is far from production ready nor optimized, so we deem ARRF itself to have minimal overhead when fragmentation is not required.

When the ARRF daemon is fragmenting resource records we see that the parallel daemon has a performance improvement of approximately 20% over TCP for all algorithms

Table 4.4: Resolution times plus and minus standard deviation with the Parallel ARRF daemon

Algorithms	Resolution Times (ms) for each Maximum Message Size (Bytes)			
	65535	1232	512	256
10ms of latency and 128 Kilobytes per second bandwidth				
Falcon-512	62.8 ± 2.2	84.7 ± 1.8	86.2 ± 2.3	89.5 ± 2.1
Dilithium2	103.1 ± 1.9	127.9 ± 1.6	132.9 ± 2.0	142.7 ± 2.0
SPHINCS+-SHA256-128s	230.7 ± 1.9	262.9 ± 2.1	279.7 ± 1.7	311.6 ± 2.1
10ms of latency and 50 Megabytes per second bandwidth				
Falcon-512	41.6 ± 2.1	62.0 ± 2.1	62.1 ± 2.3	62.2 ± 2.2
Dilithium2	41.0 ± 2.2	62.5 ± 2.2	63.0 ± 2.6	62.5 ± 2.6
SPHINCS+-SHA256-128s	42.4 ± 2.2	63.5 ± 2.2	64.4 ± 1.9	66.8 ± 2.2
100ms of latency and 50 Megabytes per second bandwidth				
Falcon-512	400.6 ± 2.0	601.1 ± 2.2	601.2 ± 2.2	601.7 ± 2.2
Dilithm2	400.9 ± 2.0	601.7 ± 2.3	601.7 ± 2.2	602.4 ± 1.9
SPHINCS+-SHA256-128s	401.5 ± 2.1	602.4 ± 1.9	603.4 ± 1.6	605.5 ± 2.4
0ms of latency and unlimited bandwidth				
Falcon-512	1.2 ± 2.4	1.5 ± 2.3	1.7 ± 2.3	1.8 ± 2.5
Dilithium2	1.2 ± 2.1	1.7 ± 2.4	1.9 ± 2.0	2.5 ± 1.9
SPHINCS+-SHA256-128s	1.4 ± 2.1	2.4 ± 1.9	3.5 ± 1.6	5.7 ± 2.4

Table 4.5: Total data transmitted when performing a DNS lookup between resolver and name server

Algorithm	Data transmitted durring DNS lookup (Bytes)			
	Standard DNS	ARRF Maximum DNS Message sizes (Bytes)		
		1232	512	256
Falcon-512	3,112	2,557	2,947	3,637
Dilithium2	8,623	8,367	9,402	11,322
SPHINCS+-SHA256-128s	26,073	26,140	29,620	36,175

and all maximum messages sizes. This is due to the parallel nature of the parallel daemon effectively only paying the latency cost once after receiving the initial response whereas TCP as a limited sized window causing the latency cost to be paid multiple times. The

sequential daemon even outperforms TCP for Falcon-512 with a maximum messages size of 1232 bytes across all tested network conditions. This is due to the Falcon-512 signed response only requiring one additional round trip to reassemble the message whereas the TCP fallback needs to receive the entire message from scratch.

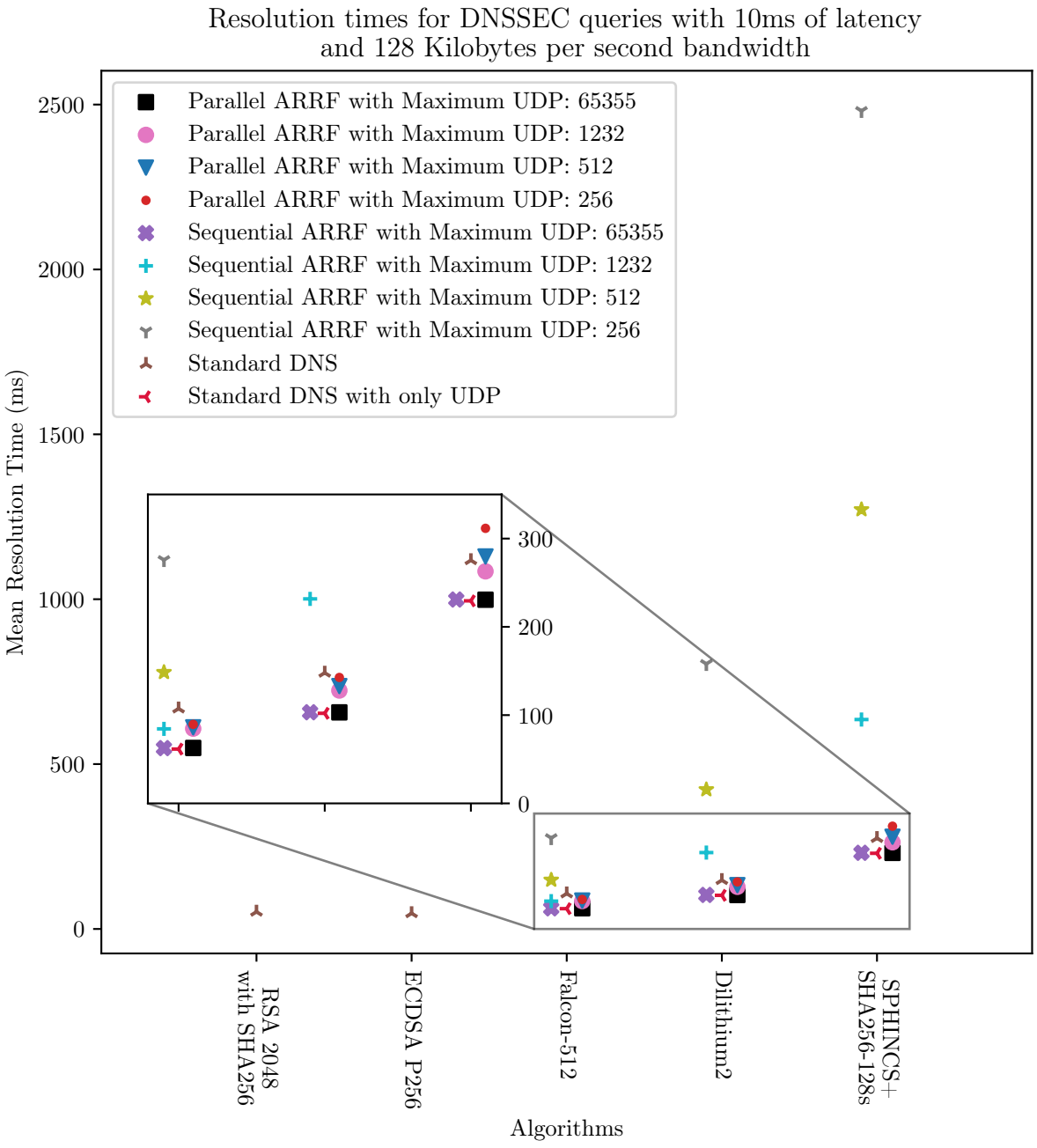


Figure 4.3: Resolution times in milliseconds with 10ms delay and 128 Kilobytes per second bandwidth

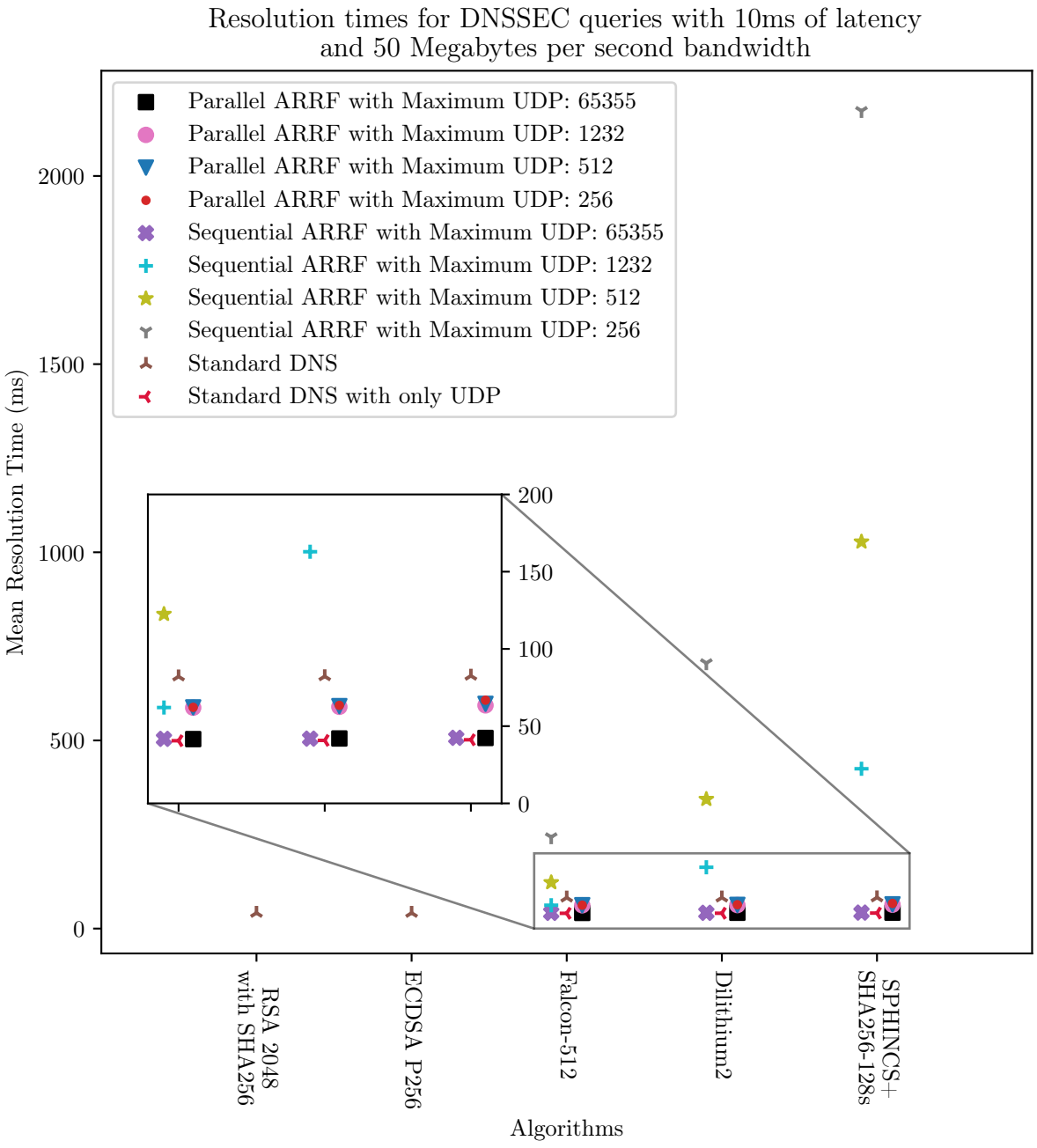


Figure 4.4: Resolution times in milliseconds with 10ms delay and 50 Megabytes per second bandwidth

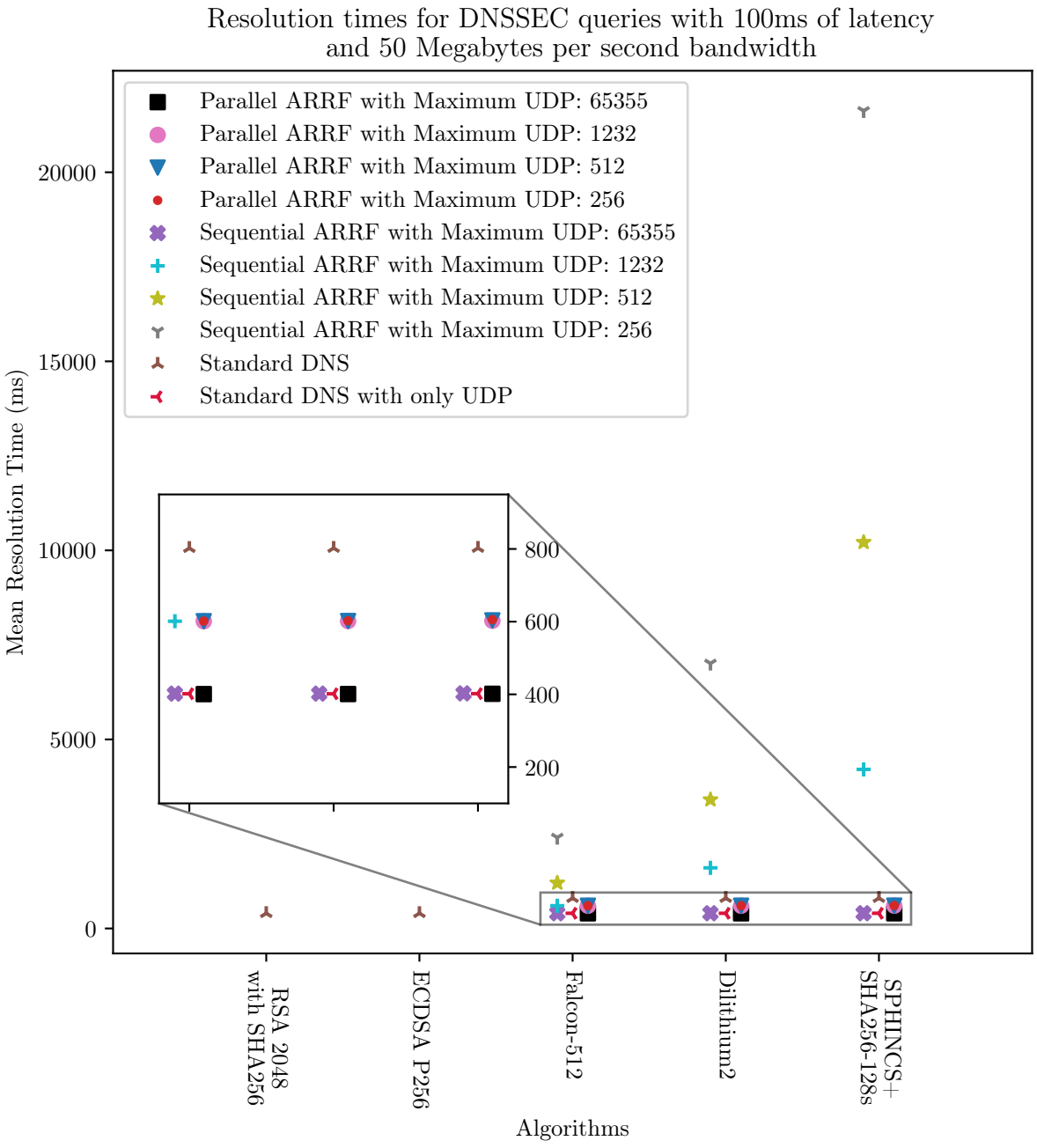


Figure 4.5: Resolution times in milliseconds with 10ms delay and 50 Megabytes per second bandwidth

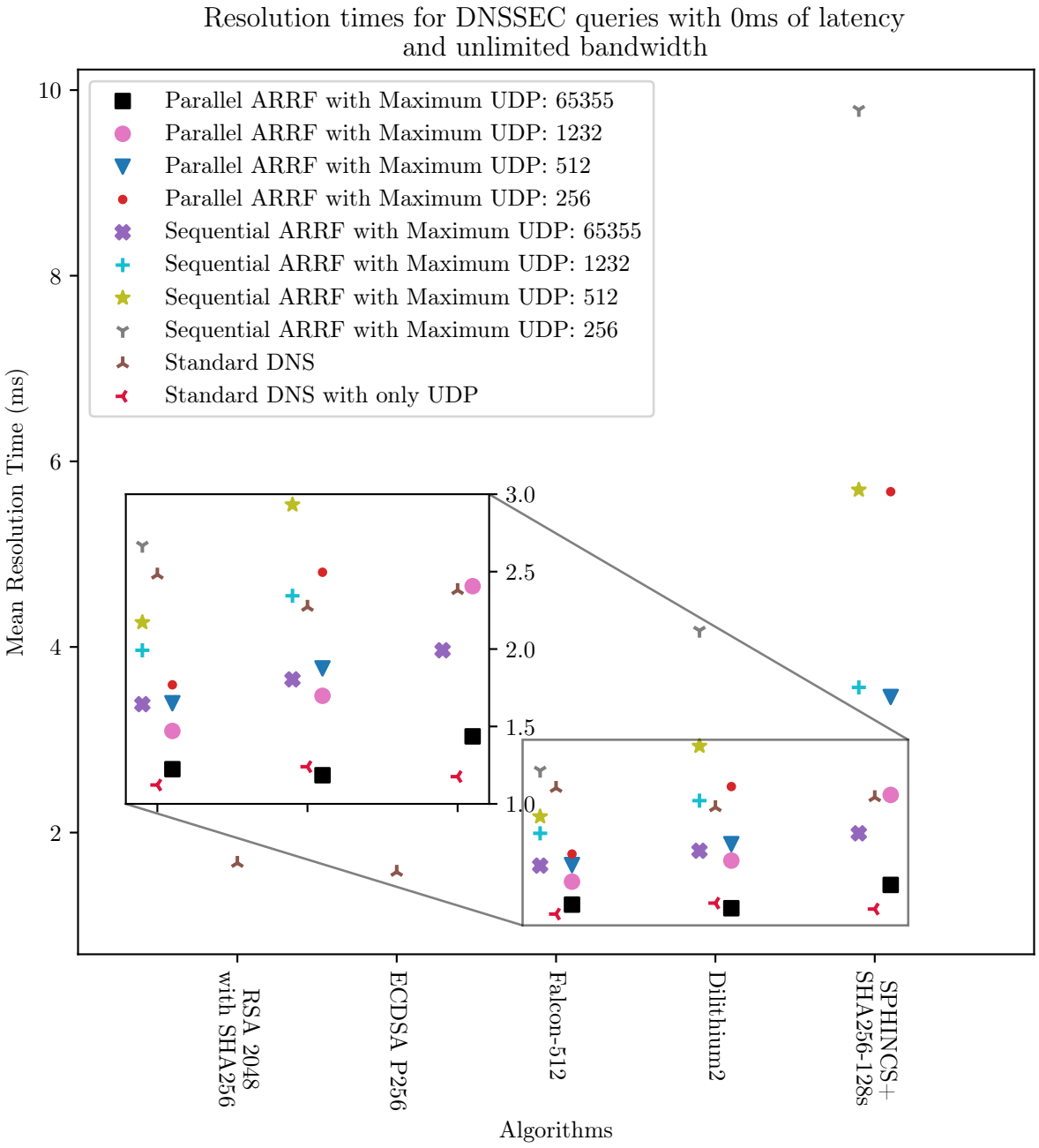


Figure 4.6: Resolution times in milliseconds with 0ms delay and unlimited bandwidth

The sequential daemon performs worse in all other cases and is greatly affected by increased latency. This is due to the sequential daemon needing to wait for each request to be fulfilled before requesting the next piece, and TCP being able to achieve some parallelism due to its sliding window. In the scenarios with latency and bandwidth restrictions, we see that as the maximum message size is reduced, Parallel ARRF scales nicely due to parallelizing the requests, whereas sequential ARRF scales roughly by the factor that the maximum message size is reduced.

When comparing algorithms, Falcon-512 comes the closest to RSA and ECDSA in all constrained network scenarios, but is still slower despite the efficient verification function. Falcon-512 is affected primarily by bandwidth and is 60% slower than RSA and 76% slower than ECDSA in the 128 kilobytes per second scenario when using Parallel ARRF. If bandwidth is not a concern, then Falcon-512 performs better, but is still 49% slower than both RSA and ECDSA in both scenarios with 50 megabytes per second bandwidth. Unsurprisingly, Dilithium2 and SPHINCS+-SHA256-128s perform far worse than Falcon-512 and the non-post-quantum algorithms; roughly 1.5 and 3 times slower than Falcon-512 when using Parallel ARRF, and even worse when using sequential ARRF.

Finally, when considering data overhead, we can see that when DNS messages sizes are at the recommended size of 1232 bytes that ARRF actually uses less data to transmit a DNS signed with Falcon-512 and Dilithium2. This is due to how DNS handles switching to TCP essentially causing the three-way TCP handshake to turn into a five-way handshake. First the resolver sends a UDP request to the name server. The name server then sends a response identical to the request and marks the response as truncated the standard TCP three-way handshake is then performed. TCP also sends acknowledgement packets for packets received and thus offsetting the extra costs imposed by fragment requests in ARRF. With these factors, combined with UDP packet headers being 12 bytes smaller than those of TCP, allows efficient communication for both Falcon-512 and Dilithium2. However, TCP becomes more data efficient compared to ARRF once many fragments are requested and sent. Due to maintaining backwards compatibility, ARRF must surround all requests and responses inside of a DNS message and all fragments inside of an RRFrag. TCP, on the other hand, is a stream and only sends a single DNS message header and sends the raw resource records themselves rather than sending the extra bytes RRFrags required. As mentioned earlier TCP sends acknowledgement packets for each TCP packet received. These acknowledgements are smaller than a UDP packet containing an ARRF request. The size difference depends on how many RRFrags are being requested, but the most common ARRF request in our experiments was 60 bytes including UDP, IP, and DNS message headers, and the largest request being 75 bytes, whereas TCP's acknowledgement packets are 52 bytes in size. If a DNS message is considerably large, such is the case with

SPHINCS+-SHA256-128s signed messages, these small data savings end up making up for wasting the initial UDP request.

4.3 Discussion

We first discuss ARRF and consider whether it is a viable solution for sending large DNS messages. We then discuss the viability of post-quantum signatures being applied to DNSSEC.

4.3.1 ARRF Performance

As can be seen in Tables 4.3 and 4.4, and Figures 4.6, 4.3, 4.4, and 4.5, Parallel ARRF is by far the most performant beating out TCP in all cases despite how many requests and responses are required to transmit the original large DNS message. Sequential ARRF when used in cases of messages being only slightly larger than what can fit in a single UDP packet also performs quite well when compared against TCP. However Parallel ARRF's performance does not come for free. On a busy resolver these parallel requests could eat up available bandwidth quite quickly and could potentially overwhelm middle boxes. We hypothesize that a production ready version of ARRF would have rate limiting to prevent this issue, and therefore performance would likely be. We also note that despite there not being considerable differences between DNS with only UDP and the ARRF daemon running but not fragmenting, that there are likely optimizations, such as multithreading, that can be made to the daemon. We also hypothesize that ARRF being integrated directly into DNS software would increase efficiency. We leave experimenting and evaluating these potential optimizations as well as evaluating window sizes as future work.

4.3.2 ARRF Backwards Compatibility

As DNS is a distributed system managed by many different entities, it is likely that there may be resolvers and/or name servers which do not understand ARRF. We now consider two scenarios: when the resolver implements ARRF, and when the resolver does not implement ARRF. We also discuss the impact ARRF has on middle boxes.

Resolver implements ARRF

When a querier which supports ARRF receives a response from a name server which does not support ARRF, it will receive a truncated DNS message with the TC flag set. It can then gracefully fallback to TCP and retry the query, thereby maintaining backwards compatibility.

Resolver does not implement ARRF

At first glance ARRF may cause issues when the querier receives a response containing an RRFrag as it will not be able to understand what an RRFrag is, nor what it should do with it. Thankfully, older resolvers ignore unknown resource record types, so it will gracefully fallback to TCP as it will see that the TC flag is set resulting in no additional round trips than if ARRF was not being used.

ARRF Middle box support

By fragmenting at the DNS level, we should ensure that the majority of middle boxes will not cause issues for ARRF. From a middle box's perspective, all messages sent using ARRF look like standard DNS messages which will not require any state to be properly routed. However, if there exist middle boxes which look inside DNS messages and view the types of the message's resource records the new RRFrag type could potentially cause those middle boxes to reject the message. Additional work is required to determine if these middle boxes are indeed a concern and, if so, how widespread these middle boxes are.

4.3.3 ARRF Security Considerations

Denial of Service Attacks

ARRF is designed to prevent DoS attacks as much as possible. Since fragments must be explicitly requested, a querier can reject any fragments it is not expecting. When combined with DNS cookies, off path attacks become infeasible. An adversary which is in-path could modify the values in response which contain RRFrags which could cause a querier to ask for fragments which do not exist. Middle boxes could also inject malicious data into individual RRFrag's FRAGDATA fields. If DNSSEC is used then this will cause the validation to eventually fail. This is acceptable as this validation failure, although denying service, is no

worse than DNS without ARRF deployed. ARRF also limits the impact of amplification DoS attacks as it restricts the response sizes and each response needs a corresponding request. If a response arrives with the wrong ID or DNS cookie, it should be discarded.

DNS Cache Poisoning

Since RRFrags themselves should not be cached, DNS cache poisoning is no more of a concern than it is in traditional DNS. If DNSSEC is used, then DNS cache poisoning is not a concern assuming a secure signature algorithm is used.

Memory Exhaustion Attacks

ARRF as specified is susceptible to memory exhaustion attacks. Although DNS cookies make this less of a concern for out of path adversaries, there is nothing stopping an on path adversary from changing the RRSIZE fields in the initial response. Since the querier uses this initial response as a map without any form of validation, an adversary could insert many RRFrags advertising they are fragments of extremely large resource records. The querier would likely then allocate enough memory to store the intermediate state until reassembly is possible. One potential solution to this would be to use some heuristics to determine if a RRFrag map makes sense. Based on what the resolver could expect to receive for a query of some form, the resolver can check to see if the response it actually received fits within those expectations. For example, if the resolver indicated that it only supported Falcon-512 signatures, it can check that the advertised sizes of the fragments are no larger than 690 bytes. We ultimately leave this issue as an avenue for future exploration.

Unreliable Networks

In this work we did not specify nor evaluate how ARRF performs when UDP packets do not reach their destination. BIND9 uses a default timeout of 800ms to determine whether it should try the request again or not, but it is unclear if that timeout duration would make sense for ARRF or not. This question must be answered before ARRF can be deployed and we leave this as an avenue for future work.

4.3.4 Comparing ARRF against previously proposed mechanisms

As discussed in Section 2.4.2, ARRF is not the first attempt at a DNS level fragmentation mechanism. Since Sivaraman's draft "DNS message fragments" [59] was not as developed

as Additional Truncated Response (ATR) [60], we will be primarily focusing on ATR in this section. ATR, Sivaraman's draft, and ARRF, all rely on DNS level fragmentation. The DNS servers are required to fragment messages and re-assemble them rather than relying on the transport layer to handle message fragmentation for them. All three mechanisms are transport layer agnostic and could therefore be used on both UDP and TCP. It may seem unclear why someone would want to run any of these mechanisms over TCP, however by doing so there is the potential for sending DNS messages larger than the 64 kilobyte maximum. ATR and Sivaraman's draft could in theory allow resource records of 64 kilobytes to be transmitted; whereas ARRF could allow for resource records of arbitrary length if RRSIZE were increased. This is due to the difference in granularity of fragmentation that the three mechanisms use. ATR and Sivaraman's draft fragment the DNS message as a whole, whereas ARRF fragments individual resource records. Although there are no resource records that require an increase to the maximum DNS message size, and therefore maximum resource record size, it is not entirely unrealistic to see this issue potentially arising.

Before being broken [12], the Rainbow [21] post-quantum signature scheme was quite appealing due to its relatively small signature sizes, however had large public keys of 161600 bytes. Since DNSKEYS are sent much less frequently than signatures, this might have been a reasonable trade off had Rainbow not been broken. It is entirely possible that a new post-quantum signature scheme is created and deemed secure which has similar signature and public key sizes. In order to fully support arbitrary sized resource records, the resource record format would need to be modified to support larger RDATA regions, and RRSIZE would need to be updated to the proper integer width.

One of the major criticisms of ATR was that since the mechanism would "blindly" send its additional message as part of its response, it would cause a flood of ICMP 'destination unreachable' packets to be created by resolvers which did not support ATR. Many implementations close their sockets immediately after receiving a response, so by the time the additional message is received the socket would no longer be accessible. This would make debugging considerably more challenging and reduce the usefulness of ICMP messages as a whole. Another issue arises with firewalls that have the policy of only receiving a single DNS message per query, and thus compounding the ICMP flood issue. ARRF does not suffer from these issues. First responses are only sent when they are explicitly queried for. A DNS server implementing ARRF will never send an additional response blindly and will never send additional messages to resolvers that do not support ARRF as they will never ask for them. Similarly, all DNS messages containing RRFrags will have an associated query and will therefore not get dropped by firewalls implementing the above policy. As ARRF does not suffer from either of those issues, there will not be a flood of ICMP packets

that caused so much concern.

ATR also requires a slight delay between the first message being sent and the trailing messages being sent in order to maintain message ordering. Receiving messages out of order is not an issue for ARRF as the requesting server will know what to expect after receiving the first message containing the "map" of the large DNS message. All responses after the first one will have been explicitly asked for and are not dependent on any other responses.

Whereas ATR is quite lightweight, ARRF does have some additional communication costs. ATR has a single round trip + the delay required to maintain message ordering, whereas ARRF has $\frac{\text{Original message size}}{\text{Maximum message size}}$ round trips. With the exception of the initial round trip these round trips can be performed in parallel thus reducing the overall resolution time as demonstrated in Figures 4.3, 4.4, 4.5, and 4.6. ARRF also requires more data to be sent, specifically as part of requesting the additional fragments. RRFrags in requests are 15 bytes in size, and the number sent depends on the number of resource records, how large they are, and how much data can fit in the maximum message size.

Sivaraman's draft uses EDNS0's OPT resource record requiring three fragmentation related options support to be assigned by ICANN. ARRF does not use the OPT pseudo-resource record and therefore does not require any options to be defined by ICANN, however a new resource record type will need to be assigned for RRFRAss.

Finally both ARRF and ATR can be implemented as a daemon on the resolver side without any changes required to the DNS software being used. This would make deployment much simpler as it would not require a DNS operator to update their resolver software and potentially have version incompatibilities. The reassembly could be performed entirely transparently to the resolver.

When considering which algorithm should be used to strengthen DNSSEC against a quantum adversary, we can see that Falcon-512 is by far the most performant whether ARRF is used or not. This is because Falcon's signature and key sizes are substantially smaller than those offered by Dilithium2 and SPHINCS+-128s. Since DNS is such a key portion of internet infrastructure, any additional overhead will be have significant impact. Assuming that Falcon's security stands up to additional scrutiny by the cryptographic community, we recommend that Falcon-512 be selected for standardization by the DNS community and that ARRF be deployed alongside Falcon to minimize the slow down caused by transmitting Falcon's keys and signatures.

Chapter 5

Future Work and Conclusion

To conclude this work we discuss the future work required to deploy ARRF and post-quantum cryptography to the DNS, and summarize our findings.

5.1 Future Work

Although ARRF appears to be a viable solution to solving DNS message fragmentation and therefore opening the door for a post-quantum secure DNS, additional work needs to be done. The backwards compatibility of ARRF needs to be further explored and evaluated; more specifically exploring if there are middle boxes which cause ARRF to fail. ARRF as specified in this work is susceptible to memory exhaustion attacks and additional work needs to be done to prevent these attacks. It is also likely that operators will want to limit the amount of concurrent requests when using Parallel ARRF and therefore research into selecting a reasonable limit must be done. In this work we provide a proof of concept daemon which transparently implements ARRF, but there may be additional concerns for integrating ARRF into the various DNS implementations and requires additional study. In this work we only considered the case when packet delivery was guaranteed. Unfortunately, this is not a guarantee we can rely on in reality and we must therefore research how ARRF behaves in unreliable networks. Finally, work needs to be done to measure any additional processing/memory overhead introduced by ARRF and whether that overhead is reasonable.

Once the above issues have been resolved, ARRF must be proposed to the IETF as an internet draft. After addressing comments from the DNS community, assuming enough

support is received, the ARRF draft must go through a 'call for adoption'. If enough support is shown during the call for adoption, a working group will be established and work on standardizing ARRF. Finally, if the working group deems it appropriate, an RFC will be published specifying the ARRF standard, and implementations should then begin adopting the mechanism.

Any proposed signature algorithm must go through a similar standardization process as ARRF will have to, however should be done independently of ARRF. Although ARRF makes sense to be deployed alongside a post-quantum signing algorithm, it is important that in the event ARRF is not standardized that the standardization of a post-quantum algorithm is not hampered.

5.2 Conclusion

Post-quantum cryptography will inevitably need to be integrated into the DNSSEC ecosystem, however it looks like it will not be as smooth of a transition as we would like. Of all the options, Falcon-512 is by far the most performant but even with Parallel ARRF is still significantly slower than currently used classical signing algorithms. There has also been work on shrinking Falcon-512 signatures significantly which will improve its performance [27]. Dilithium2 is perhaps viable as an alternative option, but considering the DNSSEC community's previous stance of "we can avoid sending large message by shaping their contents better (smaller signatures, less additional data)" [66], it is likely to receive significant resistance through the standardization process. SPHINCS+-128s is by far the worst performing of the three post-quantum options due to its slow verification and extremely large signatures which causes resolution times to explode.

Message sizes are not the only thing to consider when discussing which post-quantum signing algorithm to standardize for DNSSEC, as the security of the algorithms must also be considered. So far major attacks have been found against several algorithms fairly late in the NIST selection process. To make matters worse, those algorithms were broken with traditional computers, therefore making the attacks much more practical. Although the three selected algorithms are believed to be secure now, it remains to be seen how they hold up to additional scrutiny. It is likely that using a hybrid of a classical signing scheme and post-quantum scheme will be required to ensure that the signatures are at least as strong as what are currently standardized. This will come at a further performance cost and also increase communication sizes, and we plan to evaluate this additional cost in the future.

One option is waiting for new post-quantum signature schemes to be invented and hope that signature sizes become more reasonable. NIST is requesting additional post-quantum signature schemes be submitted for consideration for standardization [56]. However waiting for such a scheme to emerge is eating into the valuable time needed to plan and prepare for a quantum adversary. It is best that we plan for the worst case of signatures sizes not improving and be pleasantly surprised if such a scheme arises. With that in mind, we recommend Falcon-512 to be the post-quantum signing algorithm selected for DNSSEC standardization with ARRF as its delivery mechanism to achieve reasonable resolution times while providing robust fragmentation.

References

[1] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jurjen Bos, Jean-Christophe Deneuville, Arnaud Dion, Philippe Gaborit, Jérôme Lacan, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. Hamming Quasi-Cyclic (HQC) Third round version, 2021-06-06 2021.

[2] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. Status report on the first round of the NIST post-quantum cryptography standardization process, 2019-01-31 2019.

[3] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second third of the NIST post-quantum cryptography standardization process, 2022-07-5 2022.

[4] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece: conservative code-based cryptography, 2020-10-10 2020.

[5] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit Flipping Key Encapsulation Round 3 Submission, 2021-09-29 2021.

[6] Suranjith Ariyapperuma and Chris J. Mitchell. Security vulnerabilities in DNS and DNSSEC. In *The Second International Conference on Availability, Reliability and Security (ARES'07)*, pages 335–342, 2007.

[7] Derek Atkins and Rob Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, RFC Editor, August 2004.

[8] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stafan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS+ Submission to the NIST post-quantum project, v.3.1, 2022-07-10 2022.

[9] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER Algorithm Specifications and Supporitng Documentation (Version 3.02), 2021-08-04 2021.

[10] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium Algorithm Specifications and Supporitng Documentation (Version 3.1), 2021-02-8 2021.

[11] G.J. Beernink. Taking the quantum leap: Preparing DNSSEC for Post Quantum Cryptography. Master's thesis, University of Twente, February 2022.

[12] Ward Beullens. Breaking Rainbow Takes a Weekend on a Laptop. Cryptology ePrint Archive, Paper 2022/214, 2022. https://eprint.iacr.org/2022/214.

[13] Ron Bonica, Fred Baker, Geoff Huston, Bob Hinden, Ole Trøan, and Fernando Gont. IP Fragmentation Considered Fragile. RFC 8900, RFC Editor, September 2020.

[14] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A Great Multivariate Short Signature, 2020-10-21 2020.

[15] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive, Paper 2022/975, 2022. https://eprint.iacr.org/2022/975.

[16] David Conrad. Indicating resolver support of DNSSEC. RFC 3225, RFC Editor, 12 2001.

[17] Lawrence W. Conroy and Scott Petrack. The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services. RFC 2848, RFC Editor, June 2000.

[18] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. Recommendation for Stateful Hash-Based Signature Schemes, 2020-10-30 2020.

[19] Tianxiang Dai, Haya Shulman, and Michael Waidner. DNSSEC misconfigurations in popular domains. In *Cryptology and Network Security*, volume 10052, pages 651–660. Springer International Publishing, 2016.

[20] Quynh Dang. Secure hash standard (SHS). FIPS 180-4, National Institute of Standards and Technology, 2012-03-06 2012.

[21] Jintai Ding and Dieter Schmidt. Rainbow, a New Multivariable Polynomial Signature Scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 164–175. Springer International Publishing, 2005.

[22] Morris Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. FIPS 202, National Institute of Standards and Technology, 2015-08-04 2015.

[23] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (AES). FIPS 197, National Institute of Standards and Technology, 2001-11-26 2001.

[24] Donald E. Eastlake 3rd. Domain Name System Security Extensions. RFC 2535, RFC Editor, March 1999.

[25] Donald E. Eastlake 3rd and Mark P. Andrews. Domain Name System (DNS) Cookies. RFC 7873, RFC Editor, May 2016.

[26] Donald E. Eastlake 3rd, Ólafurand Guðmundsson, Paul A. Vixie, and Brian Wellingotn. Secret Key Transaction Authentication for DNS (TSIG). RFC 2845, RFC Editor, May 2000.

[27] Thomas Espitau, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Shorter Hash-and-Sign Lattice-Based Signatures. Cryptology ePrint Archive, Paper 2022/785, 2022. https://eprint.iacr.org/2022/785.

[28] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU Specification v1.2, 2020-01-10 2020.

[29] Sukhpal S. Gil, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience*, 52(1):66–114, 2022.

[30] Google. Quantum computing service. https://quantumai.google/quantum-computing-service. Accessed: 2022-07-01.

[31] Paul E. Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 8484, RFC Editor, October 2018.

[32] Paul E. Hoffman and Jakob Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, RFC Editor, August 2012.

[33] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, RFC Editor, May 2016.

[34] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, RFC Editor, May 2018.

[35] Christian Huitema, Sara Dickinson, and Allison Mankin. DNS over Dedicated QUIC Connections. RFC 9250, RFC Editor, May 2022.

[36] Geoff Huston, Jao Silva Damas, and Warren Kumari. A Root Key Trust Anchor Sentinel for DNSSEC. RFC 8509, RFC Editor, December 2018.

[37] IBM. IBM Quantum systems. https://www.ibm.com/quantum/systems. Accessed: 2022-07-01.

[38] Internet Systems Consortium. BIND 9. https://www.isc.org/bind, 2021.

[39] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Lucas De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukarev, and David Urbanik. Supersingular Isogeny Key Encapsulation, 2020-10-01 2020.

[40] Dan Kaminsky. Black ops 2008: It's the end of the cache as we know it. https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf, 2008.

[41] Olaf M. Kolkam, Matthijs Mekking, and Miek Gieben. DNSSEC Operational Practices, Version 2. RFC 6781, RFC Editor, December 2012.

[42] Annabelle Lee, Miles Smid, and Stanley Snouffer. Security requirements for cryptographic modules [includes change notices as of 12/3/2002]. FIPS 140-2, National Institute of Standards and Technology, 2001-05-25 2001.

[43] Marek Majkowski and Ólafur Guðmundsson. Deprecating the DNS ANY meta-query type. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf, 2015-03-06 2015.

[44] Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. Dns cache poisoning attack reloaded: Revolutions with side channels. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1337–1350. Association for Computing Machinery, 2020.

[45] Jiarun Mao, Michael Rabinovich, and Kyle Schomp. Assessing Support for DNS-over-TCP in the Wild. In Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser, editors, *Passive and Active Measurement*, pages 487–517, Cham, 2022. Springer International Publishing.

[46] David McGrew, Michael Curcio, and Scott Fluhrer. Leighton-Micali Hash-Based Signatures. RFC 8554, RFC Editor, April 2019.

[47] Microsoft. Azure quantum. https://azure.microsoft.com/en-ca/services/quantum/. Accessed: 2022-07-01.

[48] Paul Mockapetris. Domain names: Concepts and facilities. RFC 882, RFC Editor, November 1983.

[49] Paul Mockapetris. Domain names: Implementation specification. RFC 883, RFC Editor, November 1983.

[50] Dustin Moody, Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Jacob Alperin-Sheriff. Status report on the second round of the NIST post-quantum cryptography standardization process, 2020-07-22 2020.

[51] Moritz Müller, Jins de Jong, Maran van Heesch, Benno Overeinder, and Roland van Rijswijk-Deij. Retrofitting post-quantum cryptography in internet protocols: a case study of DNSSEC. *ACM SIGCOMM Computer Communication Review*, 50(4):49–57, 2020.

[52] Moritz Müller, Matthew Thomas, Duane Wessel, Wes Hardaker, Taejoon Chung, Willem Toorop, and Roland van Rijswijk-Deij. Roll, roll, roll your root: A comprehensive analysis of the first ever DNSSEC root KSK rollover. In *Proceedings of the ACM Internet Measurement Conference*, IMC '19, page 1–14. Association for Computing Machinery, 2019.

[53] Moritz Müller, Willem Toorop, Taejoong Chung, Jelte Jansen, and Roland van Rijswijk-Deij. The reality of algorithm agility: Studying the DNSSEC algorithm life-cycle. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 295–308. Association for Computing Machinery, 2020.

[54] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf, 2026-12-20 2016.

[55] NIST. Compliance FAQs: Federal information processing standards (FIPS). https://www.nist.gov/standardsgov/compliance-faqs-federal-information-processing-standards-fips, Nov 2019.

[56] NIST. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf, 2022-09-06 2022.

[57] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. DNS Security Introduction and Requirements. RFC 4033, RFC Editor, March 2005.

[58] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[59] Mukund Sivaraman, Shane Kerr, and Linjian Song. DNS message fragments. Internet-Draft draft-muks-dns-message-fragments-00, Internet Engineering Task Force, July 2015.

[60] Linjian Song and Shengling Wang. ATR: Additional Truncation Response for Large DNS Response. Internet-Draft draft-song-atr-large-resp-03, Internet Engineering Task Force, March 2019.

[61] Douglas Stebila and Michele Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In *Selected Areas in Cryptography (SAC) 2016*, volume 10532 of *LNCS*, pages 14–37. Springer International Publishing, 2016.

[62] Gijs van den Broek, Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC meets real world: dealing with unreachability caused by fragmentation. *IEEE Communications Magazine*, 52(4):154–160, 2014.

[63] Roland van Rijswijk-Deij, Kasper Hageman, Anna Sperotto, and Aiko Pras. The performance impact of elliptic curve cryptography on DNSSEC validation. *IEEE/ACM Transactions on Networking*, 25(2):738–750, 2017.

[64] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and its potential for DDoS attacks: A comprehensive measurement study. In *Proceedings of the ACM Internet Measurement Conference*, IMC '14, page 449–460. Association for Computing Machinery, 2014.

[65] Paul Vixie. Extension Mechanisms for DNS (EDNS0). RFC 2671, RFC Editor, August 1999.

[66] Paul Vixie. Re: [DNSOP] Call for Adoption: draft-song-atr-large-resp. https://mailarchive.ietf.org/arch/msg/dnsop/JdhkwdWT2hGzIwfVx6CrX15KCfk/, 2019.

[67] Mattäus Wander. Measurement survey of server-side DNSSEC adoption. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9, 2017.

[68] Samuel Weiler and Johan Ihren. Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC 4470, RFC Editor, April 2006.

[69] Duane Wessels, Warren Kumari, and Paul E. Hoffman. Signaling Trust Anchor Knowledge in DNS Security Extensions (DNSSEC). RFC 8145, RFC Editor, April 2017.

[70] Haisheng Yu, Yan Liu, and Guangzhou Genlian. DNS message fragments. Internet-Draft draft-hsyu-message-fragments-00, Internet Engineering Task Force, April 2022.